User and Reference Manual

Copyright © 1998–2012. Altova GmbH. All rights reserved. Use of this software is governed by and subject to an Altova software license agreement. XMLSpy, MapForce, StyleVision, SemanticWorks, SchemaAgent, UModel, DatabaseSpy, DiffDog, Authentic, AltovaXML, MissionKit, and ALTOVA as well as their logos are trademarks and/or registered trademarks of Altova GmbH. Protected by US Patent 7,200,816.

ALTOVA®

XML, XSL, XHTML, and W3C are trademarks (registered in numerous countries) of the World Wide Web Consortium; marks of the W3C are registered and held by its host institutions, MIT, INRIA, and Keio. UNICODE and the Unicode Logo are trademarks of Unicode Inc. This software contains 3rd party software or material that is protected by copyright and subject to other terms and conditions as detailed on the Altova website at http://www.altova.com/legal_3rdparty.html

Altova StyleVision 2012 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: 2012

© 2012 Altova GmbH

Table of Contents

1 Altova StyleVision 2012 3

2 About this Documentation 6

3 New Features 10

3.1 Version 2011 ... 11
3.2 Version 2010 ... 12

4 Introduction 16

4.1 What Is an SPS? ... 17
4.2 Product Features ... 19
4.3 Terminology ... 22
4.4 Setting up StyleVision ... 26
4.5 Authentic View in Altova Products ... 27

5 User Interface 30

5.1 Main Window ... 31
5.1.1 Design View ... 32
5.1.2 Output Views ... 33
5.2 Sidebars ... 35
5.2.1 Design Overview ... 38
5.2.2 Schema Tree ... 41
5.2.3 Design Tree ... 44
5.2.4 Style Repository ... 47
5.2.5 Styles ... 50
5.2.6 Properties ... 52

6 Quick Start Tutorial 58

6.1 Creating and Setting Up a New SPS ... 59
6.2 Inserting Dynamic Content (from XML Source) ... 62

6.3 Inserting Static Content ... 68
6.4 Formatting the Content ... 73
6.5 Using Auto-Calculations ... 79
6.6 Using Conditions ... 82
6.7 Using Global Templates and Rest-of-Contents ... 89
6.8 That's It! ... 93

7 Usage Overview 96

7.1 SPS and Sources ... 97
7.2 Creating the Design ... 98
7.3 XSLT and XPath Versions ... 99
7.4 Internet Explorer Compatibility ... 100
7.5 Generated Files ... 102
7.6 Catalogs in StyleVision ... 103

8 SPS File: Content 108

8.1 Inserting XML Content as Text ... 109
8.1.1 Inserting Content with a Predefined Format ... 112
8.1.2 Rest-of-Contents ... 113
8.2 User-Defined Templates ... 114
8.3 User-Defined Elements, XML Text Blocks ... 117
8.3.1 User-Defined Elements ... 118
8.3.2 User-Defined XML Text Blocks ... 119
8.4 Tables ... 120
8.4.1 Static Tables ... 122
8.4.2 Dynamic Tables ... 124
8.4.3 Tables in Design View ... 129
8.4.4 Table Formatting ... 131
8.4.5 Row and Column Display ... 135
8.4.6 CALS/HTML Tables ... 137
8.5 Lists ... 141
8.5.1 Static Lists ... 142
8.5.2 Dynamic Lists ... 144
8.6 Graphics ... 147
8.6.1 Images: URIs and Inline Data ... 148
8.6.2 Image Types and Output ... 150
8.6.3 Example: A Template for Images ... 152

8.7 Form Controls ... 153
8.7.1 Input Fields, Multiline Input Fields ... 155
8.7.2 Check Boxes ... 156
8.7.3 Combo Boxes ... 157
8.7.4 Radio Buttons, Buttons ... 160
8.8 Links ... 161
8.9 Layout Modules ... 162
8.9.1 Layout Containers ... 163
8.9.2 Layout Boxes ... 167
8.9.3 Lines ... 171
8.10 The Change-To Feature ... 173

9 SPS File: Structure 178

9.1 Schema Sources ... 180
9.1.1 DTDs and XML Schemas ... 182
9.1.2 User-Defined Schemas ... 187
9.2 Merging XML Data from Multiple Sources ... 190
9.3 Modular SPSs ... 193
9.3.1 Available Module Objects ... 195
9.3.2 Creating a Modular SPS ... 198
9.3.3 Example: An Address Book ... 202
9.4 Templates and Design Fragments ... 207
9.4.1 Main Template ... 208
9.4.2 Global Templates ... 209
9.4.3 User-Defined Templates ... 213
9.4.4 Variable Templates ... 216
9.4.5 Node-Template Operations ... 217
9.4.6 Design Fragments ... 221
9.5 XSLT Templates ... 224
9.6 Multiple Document Output ... 226
9.6.1 Inserting a New Document Template ... 227
9.6.2 New Document Templates and Design Structure ... 228
9.6.3 URLs of New Document Templates ... 229
9.6.4 Preview Files and Output Document Files ... 232
9.6.5 Document Properties and Styles ... 234

10 SPS File: Advanced Features 236

10.1 Auto-Calculations ... 237

10.1.1 Editing and Moving Auto-Calculations ... 238
10.1.2 Example: An Invoice ... 240
10.2 Conditions ... 242
10.2.1 Setting Up the Conditions ... 243
10.2.2 Editing Conditions ... 246
10.2.3 Conditions and Auto-Calculations ... 247
10.3 Grouping ... 248
10.3.1 Example: Group-By (Persons.sps) ... 251
10.3.2 Example: Group-By (Scores.sps) ... 253
10.4 Sorting ... 256
10.4.1 The Sorting Mechanism ... 257
10.4.2 Example: Sorting on Multiple Sort-Keys ... 259
10.5 Parameters and Variables ... 262
10.5.1 User-Declared Parameters ... 263
10.5.2 Parameters for Design Fragments ... 265
10.5.3 SPS Parameters for Sources ... 268
10.5.4 Variables ... 269
10.6 Table of Contents, Referencing, Bookmarks ... 271
10.6.1 Marking Items for TOC Inclusion ... 274
Structuring the Design in Levels ... 276
Creating TOC Bookmarks ... 279
10.6.2 Creating the TOC Template ... 282
Reflevels in the TOC Template ... 284
TOC References: Name, Scope, Hyperlink ... 285
Formatting TOC Items ... 286
10.6.3 Example: Simple TOC ... 287
10.6.4 Example: Hierarchical and Sequential TOCs ... 290
10.6.5 Auto-Numbering ... 294
10.6.6 Text References ... 298
10.6.7 Bookmarks and Hyperlinks ... 300
Inserting Bookmarks ... 301
Defining Hyperlinks ... 304
10.7 Example: Multiple Languages ... 308

11 SPS File: Presentation 312

11.1 Predefined Formats ... 313
11.2 Output Escaping ... 315
11.3 Value Formatting (Formatting Numeric Datatypes) ... 317
11.3.1 The Value Formatting Mechanism ... 318

11.3.2 Value Formatting Syntax ... 321
11.4 Working with CSS Styles ... 326
11.4.1 External Stylesheets ... 328
11.4.2 Global Styles ... 331
11.4.3 Local Styles ... 334
11.4.4 Setting Style Values ... 336
11.4.5 Style Properties Via XPath ... 338
11.4.6 Composite Styles ... 341

12 SPS File: Additional Functionality 346

12.1 Altova Global Resources ... 347
12.1.1 Defining Global Resources ... 348
Files ... 350
Folders ... 353
Copying Configurations ... 354
12.1.2 Using Global Resources ... 355
Assigning Files and Folders ... 356
Assigning Databases ... 359
Changing Configurations ... 360
12.2 Unparsed Entity URIs ... 361
12.3 New from XSLT, XSL-FO or FO File ... 363
12.4 User-Defined XPath Functions ... 366
12.4.1 Defining an XPath Function ... 369
12.4.2 Reusing Functions to Locate Nodes ... 372
12.4.3 Parameters in XPath Functions ... 373
Parameters and Sequences ... 375
Parameters and Nodes ... 379
12.5 Working with Dates ... 381
12.5.1 Formatting Dates ... 382
12.6 Using Scripts ... 385
12.6.1 Defining JavaScript Functions ... 386
12.6.2 Assigning Functions as Event Handlers ... 387
12.6.3 External JavaScript Files ... 388
12.7 HTML Import ... 390
12.7.1 Creating New SPS via HTML Import ... 391
12.7.2 Creating the Schema and SPS Design ... 393
12.7.3 Creating Tables and Lists as Elements/Attributes ... 395
12.7.4 Generating Output ... 397
12.8 ASPX Interface for Web Applications ... 398

12.8.1 Example: Localhost on Windows 7 ... 400

13 Automated Processing 404

13.1 Command Line Interface: StyleVisionBatch ... 405
13.1.1 StyleVisionBatch Syntax ... 406
13.1.2 StyleVisionBatch Examples ... 409
13.2 Using AltovaXML ... 410
13.2.1 XSLT 1.0 CLI Transformations ... 411
13.2.2 XSLT 2.0 CLI Transformations ... 412
13.3 How to Automate Processing ... 413
13.3.1 Creating Batch Files ... 414
13.3.2 Automating with Scheduled Tasks (Windows XP) ... 415
13.3.3 Automating with Scheduled Tasks (Windows Vista) ... 418

14 Reference 424

14.1 Toolbars ... 425
14.1.1 Formatting ... 428
14.1.2 Table ... 429
14.1.3 Insert Design Elements ... 431
14.1.4 Design Filter ... 433
14.1.5 Global Resources ... 434
14.1.6 Standard ... 435
14.2 Design View ... 436
14.2.1 Symbols ... 437
14.2.2 Edit XPath Expression ... 441
14.3 File Menu ... 444
14.3.1 New ... 445
14.3.2 Open, Reload, Close, Close All ... 452
14.3.3 Save Design, Save All ... 457
14.3.4 Save As ... 462
14.3.5 Save Generated Files ... 463
14.3.6 Web Design ... 464
14.3.7 Properties ... 465
14.3.8 Print Preview, Print ... 466
14.3.9 Most Recently Used Files, Exit ... 467
14.4 Edit Menu ... 468
14.4.1 Undo, Redo, Select All ... 469
14.4.2 Find, Find Next, Replace ... 470

14.4.3 Stylesheet Parameters ... 472
14.4.4 Collapse/Expand Markup ... 473
14.5 View Menu ... 474
14.5.1 Toolbars and Status Bar ... 475
14.5.2 Design Sidebars ... 476
14.5.3 Design Filter, Zoom ... 477
14.6 Insert Menu ... 478
14.6.1 Contents ... 479
14.6.2 Rest of Contents ... 480
14.6.3 Form Controls ... 481
14.6.4 Auto-Calculation ... 482
14.6.5 Paragraph, Special Paragraph ... 484
14.6.6 Image ... 485
14.6.7 Horizontal Line ... 486
14.6.8 Table ... 487
14.6.9 Bullets and Numbering ... 488
14.6.10 Bookmark ... 491
14.6.11 Hyperlink ... 492
14.6.12 Condition, Output-Based Condition ... 494
14.6.13 Template ... 496
14.6.14 User-Defined Template ... 497
14.6.15 Variable Template ... 498
14.6.16 Design Fragment ... 499
14.6.17 Layout Container, Layout Box, Line ... 500
14.6.18 Table of Contents ... 501
14.6.19 New Document ... 502
14.6.20 User-Defined Item ... 503
14.7 Enclose With Menu ... 504
14.7.1 Template ... 505
14.7.2 User-Defined Template ... 506
14.7.3 Variable Template ... 507
14.7.4 Paragraph, Special Paragraph ... 508
14.7.5 Bullets and Numbering ... 509
14.7.6 Bookmarks and Hyperlinks ... 510
14.7.7 Condition, Output-Based Condition ... 511
14.7.8 TOC Bookmarks and TOC Levels ... 513
14.7.9 New Document ... 514
14.7.10 User-Defined Element ... 515
14.8 Table Menu ... 516
14.8.1 Insert Table, Delete Table ... 517

14.8.2 Add Table Headers, Footers ... 518
14.8.3 Append/Insert Row/Column ... 519
14.8.4 Delete Row, Column ... 520
14.8.5 Join Cell Left, Right, Below, Above ... 521
14.8.6 Split Cell Horizontally, Vertically ... 522
14.8.7 View Cell Bounds, Table Markup ... 523
14.8.8 Table Properties ... 524
14.8.9 Edit CALS/HTML Tables ... 525
14.8.10 Vertical Alignment of Cell Content ... 526
14.9 Properties Menu ... 527
14.9.1 Edit Bullets and Numbering ... 528
14.9.2 Predefined Value Formatting Strings ... 529
14.10 Tools Menu ... 531
14.10.1 Spelling ... 532
14.10.2 Spelling Options ... 533
14.10.3 Global Resources ... 536
14.10.4 Active Configuration ... 537
14.10.5 Customize ... 538
14.10.6 Restore Toolbars and Windows ... 543
14.10.7 Options ... 544
14.11 Window Menu ... 546
14.12 Help Menu ... 547
14.12.1 Table of Contents, Index, Search ... 548
14.12.2 Activation, Order Form, Registration, Updates ... 549
14.12.3 Other Commands ... 550

15 Appendices 552

15.1 XSLT Engine Information ... 553
15.1.1 XSLT 1.0 Engine: Implementation Information ... 554
15.1.2 XSLT 2.0 Engine: Implementation Information ... 556
General Information ... 557
XSLT 2.0 Elements and Functions ... 559
15.1.3 XPath 2.0 and XQuery 1.0 Functions ... 560
General Information ... 561
Functions Support ... 563
15.1.4 Extensions ... 566
Java Extension Functions ... 567
.NET Extension Functions ... 575
MSXSL Scripts for XSLT ... 581

Altova Extension Functions ... *584*
15.2 Technical Data ... 588
15.2.1 OS and Memory Requirements ... 589
15.2.2 Altova XML Parser ... 590
15.2.3 Altova XSLT and XQuery Engines ... 591
15.2.4 Unicode Support ... 592
Windows XP ... *593*
Right-to-Left Writing Systems ... *594*
15.2.5 Internet Usage ... 595
15.3 License Information ... 596
15.3.1 Electronic Software Distribution ... 597
15.3.2 Software Activation and License Metering ... 598
15.3.3 Intellectual Property Rights ... 599
15.3.4 Altova End User License Agreement ... 600

Index **611**

Chapter 1

Altova StyleVision 2012

1 Altova StyleVision 2012

Altova® StyleVision® 2012 Basic Edition is an application for graphically designing and editing StyleVision Power Stylesheets. 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. (Both XSLT 1.0 and XSLT 2.0 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.

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

Altova website: Stylesheet Designer, XSLT Designer

Last updated: 02/17/2012

Chapter 2

About this Documentation

2 About this Documentation

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

The user manual is organized into the following sections:

- An introduction, which explains what an SPS is and introduces the main features and concepts of StyleVision.
- A description of the user interface, which provides an overview of the StyleVision GUI.
- A tutorial section, which is a hands-on exercise to familiarize you with StyleVision features.
- Usage Overview, which describes usage at a high level: for example, schema sources used to create an SPS, the broad design process, Authentic View deployment, and projects.
- SPS File Content, which explains how static (stylesheet-originated) and dynamic (XML document-originated) components are created and edited in the SPS.
- SPS File Structure, which shows how an SPS file can be structured and modularized, and describes the handling of StyleVision's templates.
- SPS File Advanced Features, which describes advanced design features, such as the automatic generation of calculations, the setting up of conditions, grouping and sorting on user-defined criteria, and how to build tables of contents and cross-references in the output document.
- SPS File Presentation, which explains how SPS components are formatted and laid out.
- SPS File Additional Editing Functionality, which describes a range of additional features that can make your SPS more powerful. These features include: global resources for leveraging functionality in other Altova products, additional validation, scripts, and variables and parameters.
- A reference section containing descriptions of all symbols and commands used in StyleVision.
- Appendices 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 and Usage Overview sections first in order to get an overview of StyleVision features and general usage. Doing the tutorial next would provide hands-on experience of creating an SPS. The SPS File sections (SPS File Content, SPS File Structure, SPS File Advanced Features, SPS File Presentation, SPS File Additional Functionality) provide detailed descriptions of how to use various StyleVision features. For subsequent reference, the Reference section provides a concise description of all toolbar icon, design symbols, and menu commands, organized according to toolbar and menu. The Command Line Interface: StyleVisionBatch section provides information about calling StyleVision from the command line.

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

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

- *(My) Documents folder:* The My Documents folder of Windows XP is the Documents folder of Windows Vista and Windows 7. It is located by default at the following

 © 2012 Altova GmbH

respective locations. Example files are usually located in a sub-folder of the (My) Documents folder.

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

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

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

Support options

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

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

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

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

© 2012 Altova GmbH

Chapter 3

New Features

3 New Features

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

- Composite Styles: Composite Styles enable you to combine multiple CSS style declarations in a single style rule and to apply this combined style rule to the most frequently used design components.
- Switching among external CSS Stylesheets: You can choose between using all rules in all the external CSS stylesheets associated with the SPS (in their cascading order of precedence), or the rules in one of the external CSS stylesheets.
- New SPS based on an SPS Module: A new SPS file can created that will contain an empty main template and the selected SPS file as a module.
- Find in Projects: Enables project files and folders to be located quickly by searching for text strings in their names.

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

- Bug fixes

 © 2012 Altova GmbH

3.1 Version 2011

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

- CALS/HTML Tables: The XML Tables feature of earlier versions has been improved and renamed to CALS/HTML Tables. If a CALS or HTML table structure is defined in the SPS design's DTD or schema, you can specify in the design that CALS/HTML tables should be processed. These table structures, if present in the XML instance file, will then be correctly sent to the output as tables.
- New from XSLT: An SPS can be created from an XSLT-for-HTML or an XSLT-for-FO. Template structure and styling in the XSLT will be created in the SPS. You can then modify the SPS components and add content and formatting to the SPS.

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

- Multiple Document Output: The output generated by the SPS can be designed to be split into multiple documents. In the design, New Document templates are created and content placed in them. Each New Document template generates a separate document in the output.
- User-Defined XPath Functions: The user can define XPath functions which can be used anywhere in the document where XPath functions may be used.
- Images from inline data: Images can be generated from Base-16 and Base-64 encoded text in the XML document. Consequently, images can be stored directly in the source XML document as text. An SPS can now decode such text and render the image.
- ASPX Interface for Web Applications: With this feature, HTML web pages can be quickly updated. StyleVision generates, from an SPS, all the files necessary for an ASPX application. When the web page (a `.aspx` file) is refreshed, the source data (including any updates) is dynamically transformed via XSLT to the web page.
- Combo Boxes: The design capability for combo boxes has been extended. The visible values in the dropdown list of the combo box as well as the corresponding values placed in the XML file can be separately specified.

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

- Internet Explorer 9 Compatibility: StyleVision offers support for Internet Explorer 9 (IE 9). CSS styles and HTML5 elements that are supported by IE 9 are supported in the SPS design interface. HTML Preview will show the IE 9 rendering of the output document. In the Properties dialog, you can modify the IE-compatibility of the SPS.
- Improved support for image formats: Better support for the TIFF, SVG, and JPEG XR formats.

3.2 Version 2010

Version 2010 Release 1

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

- Layout Containers: A Layout Container is a block in which Design Elements can be laid out and absolutely positioned within the block.
- Blueprints: Within a Layout Container an image of a form can be used as an underlay blueprint for the design. With the help of a blueprint, an existing design can be reproduced accurately.
- User-Defined Templates: A template can be generated for a sequence of items by an XPath expression you specify. These items may be atomic values or nodes. An XPath expression enables the selection of nodes to be more specific, allowing conditions and filters to be used for the selection. Furthermore, templates can be built for atomic values, thus enabling structures to be built that are independent of the schema structure. *Older features affected:* Variable Iterators, which were used to create a template for a variable, now create a variable on a node template and then a User-Defined template for that variable.
- User-Defined Elements: This feature is intended to enable presentation language elements (such as HTML, XSLT, and XSL-FO) to be freely inserted at any location in the design.
- User-Defined XML Text Blocks: XML Text blocks can be freely inserted at any location in the design, and these blocks will be created at that location in the generated XSLT stylesheet.
- XSLT Templates: XSLT files can be imported into the generated stylesheets. If a node in the XML instance document is matched to a template in the imported XSLT file and no other template takes precedence over the imported template, then the imported template will be used. Additionally, named templates in the imported XSLT file can be called from within the design.
- Variables: A variable can now be declared on a template and take a value that is specified with an XPath expression. Previously, the value of a variable was limited to the selection of the node on which it was created. Variables in the 2010 version allow any XPath expression to be specified as the value of the variable. *Older features affected:* Variables and Variable Iterators. Variables from older versions are now created on the relevant template and are given a value that selects the same template. Variable Iterators are replaced with a combination of a Variable and a User-Defined Template; see User-Defined Templates below.
- Inserting Design Elements: Design Elements (paragraphs, lists, images, etc) can be inserted first, and an XML node from the schema tree assigned to the Design Element afterwards. This is in addition to the existing mechanism by which a schema nodes is dragged into the design and a Design Element created for it.
- Hide Markup in Design View: Markup tags in Design View can be hidden and collapsed, thus freeing up space in Design View.
- Disable output escaping: A setting that defines whether text output will be escaped or not. A character is said to be escaped when it is written as a character entity (such as `&` or `A`). This feature is useful when outputting text that contains program code.
- Default length units: can be specified in the Options dialog (**Tools | Options**).
- XHTML output: When XHTML is specified as the HTML output preference in the document's properties (**File | Properties**), an XHTML document is generated for the HTML output.
- Printout of Design: The design in Design View can be printed with or without tags.

 © 2012 Altova GmbH

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

- Enterprise and Professional editions are each available as separate 64-bit and 32-bit applications.
- Parameters for Design Fragments allow design fragments to be used with different parameter values for each usage instance. A different parameter value can be assigned to a design fragment at each location where the design fragment is used in the SPS.
- Layout Boxes and Lines can be moved and resized using the keyboard.
- Templates around table rows or columns can be added or deleted without modifying the content or formatting of the row or column involved.
- Text in tables and in layout boxes can be rotated clockwise or anti-clockwise so that it is vertical.
- Filters can be set on global templates where these are used in the main template.
- Design fragments can be dragged from the Schema Tree, in addition to being available in the Design Tree.

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

- Value Formatting (Formatting Numeric Datatypes): The earlier Input Formatting mechanism has been extended to enable—only in the Enterprise Edition—the formatting of Inline XBRL values when they are output in an (X)HTML report. The older Input Formatting feature remains unchanged but has been renamed to Value Formatting.
- Global templates can now be created for any node or type in the schema. In earlier versions of StyleVision, global templates could only be created for global elements and global types. They can now be created on any node or type, and even for any item returned by an XPath expression.

Chapter 4

Introduction

4 Introduction

This section introduces you to **Altova® StyleVision® 2012**. It consists of the following sub-sections:

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

© 2012 Altova GmbH

4.1 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 is the data in one XML document. The static content is content entered directly in the SPS. Dynamic content can be included in the design either as straight text or within components such as input fields, combo boxes, and tables. Additionally, dynamic content can be manipulated (using Auto-Calculations) and can be displayed if certain conditions in the source document are fulfilled. Different pieces of content can be placed at various and multiple locations in the SPS. Also, the SPS can contain various other components, such as images, hyperlinks, and JavaScript functions. Each component of the SPS can then be formatted for presentation as required.

The SPS and XSLT stylesheets
After you have completed designing the SPS, you can generate XSLT stylesheets based on the design you have created. StyleVision supports both XSLT 1.0 and XSLT 2.0, and from a single SPS, you can generate XSLT stylesheets for HTML, RTF, XSL-FO, and Word 2007-and-higher output (*XSL-FO and Word 2007-and-higher in Enterprise edition only; RTF in Enterprise and Professional Editions; in Basic Edition only HTML output is supported*). The generated XSLT stylesheets can be used in external transformations to transform XML documents based on the same schema as the SPS from which the XSLT stylesheet was generated. For more information about procedures used with XSLT stylesheets, see the section Generated Files.

The SPS and output
You can also use StyleVision to directly generate output (*HTML, RTF, XSL-FO, and PDF in Enterprise Edition; HTML in Professional and BasicEditions*). The tabs for Output Views display the output for the active SPS document directly in the StyleVision GUI. The required output can also be generated to file: (i) from within the GUI via the **File | Save Generated Files** command; or (ii) by invoking StyleVision via the command line.

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

© 2012 Altova GmbH

4.2 Product Features

The main product features of StyleVision are listed below:

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

- Enterprise and Professional editions are each available as separate 64-bit and 32-bit applications.
- StyleVision functionality can be called from the command line.

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

- Altova Global Resources can be used to locate source files such as schema, XML, and CSS. The Global Resources mechanism enables faster and better development and testing by allowing developers to quickly change source data and to use the functionality of other Altova applications from within StyleVision.
- HTML documents can be converted to XML.

Interface
Given below are some general GUI features:

- Multiple SPS designs can be open simultaneously, with one being active at any given time. Each SPS design is shown in a separate tab.
- Template filters allow you to customize the display of the design document. With this feature you can disable the display of templates that are not currently being edited, thus increasing editing efficiency.
- Hide Markup in Design View: Markup tags in Design View can be hidden and collapsed, thus freeing up space in Design View.
- While designing the SPS, output views 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:

- Both XSLT versions (1.0 and 2.0) are supported. XSLT 2.0 provides powerful data access and manipulation features.
- In the Enterprise and Professional Editions, multiple output formats (HTML) are generated from a single SPS design.
- Both XSLT files and output files can be generated and saved, either directly from within the GUI or by calling StyleVision from the command line.
- ASPX Interface for Web Applications: With this feature, HTML web pages can be quickly updated. StyleVision generates, from an SPS, all the files necessary for an ASPX application. When the web page (a `.aspx` file) is refreshed, the source data (including any updates) is dynamically transformed via XSLT to the web page.

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

- The SPS can contain static text, which you enter in the SPS, and dynamic text, which is

selected from the source document.

- Dynamic content is inserted in the design by dragging-and-dropping nodes from the schema source. Design Elements (paragraphs, lists, images, etc) can also be inserted first, and an XML node from the schema tree can be assigned to the Design Element afterwards.
- Dynamic content can be inserted as text, or in the form of a data-entry device (such as an input field or combo box).
- The structure of the design is specified and controlled in a single main template. This structure can be modified by optional templates for individual elements—known as global templates because they can be applied globally for that element.
- Global templates can also be created for individual datatypes, thus enabling processing to be handled also on the basis of types.
- Multiple Document Output: The output generated by the SPS can be designed to be split into multiple documents. In the design, New Document templates are created and content placed in them. Each New Document template generates a separate document in the output.
- User-Defined Templates: A template can be generated for a sequence of items by an XPath expression you specify. These items may be atomic values or nodes. An XPath expression enables the selection of nodes to be more specific, allowing conditions and filters to be used for the selection.
- User-Defined Elements: This feature is intended to enable presentation language elements (such as HTML, XSLT, and XSL-FO) to be freely inserted at any location in the design.
- User-Defined XML Text Blocks: XML Text blocks can be freely inserted at any location in the design, and these blocks will be created at that location in the generated XSLT stylesheet.
- Design Fragments enable the modularization and re-use of templates within an SPS, and also across multiple SPSs (*see modular SPSs*), in a manner similar to the way functions are used.
- SPS modules can be added to other SPS modules, thus making objects defined in one SPS module available to other modules. This enables re-use of module objects across multiple SPSs and makes maintenance easier.
- XSLT Templates: XSLT files can be imported into the generated stylesheets. If a node in the XML instance document is matched to a template in the imported XSLT file and no other template takes precedence over the imported template, then the imported template will be used. Additionally, named templates in the imported XSLT file can be called from within the design.
- New from XSLT: An SPS can be created from an XSLT-for-HTML or an XSLT-for-FO. Template structure and styling in the XSLT will be created in the SPS. You can then modify the SPS components and add content and formatting to the SPS.
- User-Defined XPath Functions: The user can define XPath functions which can be used anywhere in the document where XPath functions may be used.
- Layout Containers: A Layout Container is a block in which Design Elements can be laid out and absolutely positioned within the block.
- Blueprints: Within a Layout Container an image of a form can be used as an underlay blueprint for the design. With the help of a blueprint, an existing design can be reproduced accurately.
- A common feature of XML documents is the repeating data structure. For example, an office department typically has several employees. The data for each employee would be stored in a data structure which is repeated for each employee. In the SPS, the processing for each such data structure is defined once and applied to each relevant node in turn (the employee node in our example).
- Multiple tables of contents can be inserted in XSLT 2.0 SPSs.
- Repeating data structures can also be inserted as dynamic tables. This provides looping in a structured, table format, with each loop through the data structure producing a row (or, if required, a column) of the table.

 © 2012 Altova GmbH

- A repeating element can be sorted on one or more sort-keys you select, and the sorted element set is sent to the output (HTML).
- Variables: A variable can now be declared on a template and take a value that is specified with an XPath expression. Previously, the value of a variable was limited to the selection of the node on which it was created. Variables in the 2010 version allow any XPath expression to be specified as the value of the variable.
- Nodes can be grouped on the basis of common data content (for example, the common value of an attribute value) and their positions.
- The conditional templates feature enables one of a set of templates to be processed according to what conditions in the XML document or system environment are fulfilled. This enables processing that is conditional on information contained in the source document or that cannot be known to the SPS document creator at the time of creation (for example, the date of processing). The available conditions are those that can be tested using XPath 1.0 or XPath 2.0 expressions.
- Auto-Calculations enable you to manipulate data from the source document/s and to display the result. This is useful, when you wish to perform calculations on numbers (for example, sum the prices in an invoice), manipulate strings (for example, change hyphens to slashes), generate content, etc. The available manipulations are those that can be effected using XPath 1.0 or XPath 2.0 expressions. Native Java and .NET functions can be used in the XPath expressions of Auto-Calculations.
- Images can be inserted in the design. The URI for the image can be static (entered in the SPS), or dynamic (taken from a node in the source document), or a combination of both static and dynamic parts.
- Images from inline data: Images can be generated from Base-16 and Base-64 encoded text in the XML document. Consequently, images can be stored directly in the source XML document as text. An SPS can now decode such text and render the image.
- Two types of lists can be created: static and dynamic. In a static list, each list item is defined in the SPS. In a dynamic list, a node is created as a list item; the values of all instances of that node are created as the items of the list.
- Static and dynamic links can be inserted in the design. The target URI can be static (entered in the SPS), or dynamic (taken from a node in the source document), or a combination of both static and dynamic parts.
- Static bookmarks can be inserted. These serve as anchors that can be linked to with a hyperlink.
- Parameters can be declared globally for the entire SPS. A parameter is declared with a name and a string value, and can be used in XPath expressions in the SPS. The parameter value you declare is the default value and can be overridden by a value passed from the command line.
- With the Input Formatting 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.
- JavaScript functions can be used in the SPS to provide user-defined functionality for Authentic View and HTML output.
- A number of predefined HTML formats are available via the GUI and can be applied to individual SPS components.
- A large number of CSS text formatting and layout properties can be applied to individual SPS components via the Styles sidebar.
- Additionally, CSS styles can be defined for HTML selectors at the global level of an SPS and in external CSS stylesheets. These style rules will be applied to HTML output, thus providing considerable formatting and layout flexibility.
- Styles can also be assigned using XPath expressions. This enables style property values to be selected from XML documents and to set property values conditionally.

4.3 Terminology

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

- Altova product-related terms
- General XML terms and concepts
- XSLT and XPath terms
- StyleVision-specific terms

Note: If a link below points to a term already in the viewport, the screen display will not change when the link is clicked; in such cases, look for the target term in the current display.

Altova product-related terms

A list of terms that relate to Altova products.

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

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

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

General XML terms

Definitions of certain XML terms as used in this documentation.

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

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

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

XSLT and XPath terms

 © 2012 Altova GmbH

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

absolute XPath A path expression that starts at the root node of the tree containing the context node. In StyleVision, when entering path expressions in dialogs, the expression can be entered as an absolute path if you check the Absolute XPath check box in the dialog. If this check box is unchecked, the path is relative to the context node.

context item / context node 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).

current node The current node is the node being currently processed. The current node is the same as the context node in expressions that do not have sub-expressions. But where there are sub-expressions, the context node may change. Note that the `current()` function is an XSLT function, not an XPath function, and cannot therefore be used in StyleVision's Auto-Calculations and Conditional Templates. To select the current node in an expression use the `for` expression of XPath 2.0.

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, 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, 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 can be defined only for global elements.

global template A global template may be defined for a global element. Once defined, a global template can be used for that element wherever that element occurs in the document. Alternatively to the global template, processing for a global element may be defined in a local template.

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. The local template applies to that particular occurrence of the element in the main template. Instead of the local template, a global template can be applied to a given occurrence of an element in the main template.

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

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

output 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 or a text string; or (iii) a predefined format (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

 © 2012 Altova GmbH

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). 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.

4.4 Setting up StyleVision

Altova StyleVision runs on Windows XP, Windows Vista, and Windows 7. After downloading StyleVision from the Altova website, double-click the executable (`.exe`) file to run the setup program. The setup program will install StyleVision at the desired location. The Altova XSLT Engines (1.0 and 2.0) are built into StyleVision and are used for all internal transformations. You, therefore, do not need to install an XSLT Engine additionally to your StyleVision installation.

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. Internet Explorer 9 is supported.

 © 2012 Altova GmbH

4.5 Authentic View in Altova Products

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

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

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

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

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

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

Note: StyleVision Enterprise Edition supports the Enterprise Edition of Authentic View, whereas StyleVision Professional Edition supports the Community Edition of Authentic View.

Chapter 5

User Interface

5 User Interface

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

- A **menu bar**. Click on a menu to display the items in that menu. All menus and their items are described in the User Reference section. The menu bar also contains the Minimize, Restore, and Close Active Document buttons.
- A **toolbar area**. The various toolbars and the command shortcuts in each toolbar are described in the User Reference section.
- A tabbed **Main Window**, which displays one or more open SPS documents at a time. In this window, you can edit the design of the SPS and preview the XSLT stylesheets and output.
- The **Design Sidebars**—the Design Overview, Schema Tree, Design Tree, Style Repository, Styles, Properties 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 and Design sidebars are described in more detail in the sub-sections of this section.

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

 © 2012 Altova GmbH

5.1 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**).

Document views

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

- Design View, in which you design the SPS and edit JavaScript functions for use in that SPS. The view can be toggled between the design document and the JavaScript Editor by clicking the dropdown menu arrow and selecting Design or JavaScript, as required.
- Output Views (HTML 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.

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. In the JavaScript Editor you can create and edit JavaScript functions which then become available in the GUI for use in the SPS. To switch to the JavaScript Editor, 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, which are available as toolbar icons. These display filters will help you optimize and switch between different displays of your SPS.

Displaying markup tags

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

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

 © 2012 Altova GmbH

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 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**) and text from it can be copied to the clipboard (with **Edit | Copy**).

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

HTML preview
HTML preview displays the output produced by transforming the Working XML File with the XSLT-for-HTML. The output is generated afresh each time 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 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 since a temporary non-editable XML file is automatically generated when the DB is loaded and this XML file is used as the Working XML File.

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

 © 2012 Altova GmbH

5.2 Sidebars

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

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

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, select Auto Hide; (to display the Menu, right-click the title bar of the window or click the Menu button in the title bar of the docked window).

The window will be auto-hidden.

To switch the Auto-Hide feature for a particular window off, place the cursor over the tab so that the window rolls out, and then click the Auto Hide button (*screenshot below*). Alternatively, in the Menu, deselect Auto Hide; (to display the Menu, right-click the title bar of the window or click the Menu button in the title bar of the window).

 © 2012 Altova GmbH

Note: When the Auto-Hide feature of a sidebar window is off, the drawing pin icon of that window points downwards; when the feature is on, the drawing pin icon points left.

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

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

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

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 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.

 © 2012 Altova GmbH

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

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

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

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

Note: The Design Overview sidebar provides an overview of the CSS files, enabling you to manage CSS files at the file level. The various CSS rules inside the CSS files, however, are listed in the Style Repository sidebar.

Parameters
The Parameters section lists the global parameters in the SPS. You can add new parameters

using the **Add New Parameter** command at the bottom of the section. Double-clicking the parameter name or value enables you to edit the name or value, respectively. To remove a parameter, select the parameter and then click the **Remove** command in the context menu.

XSLT Files

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

© 2012 Altova GmbH

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.
- Insert and and edit user-defined XPath functions for the SPS.

Root elements

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

To select the root elements for a schema, do the following: Click the **Select** button at the right of the `Root Elements` item. This pops up the Select Root Elements dialog (*screenshot below*), in which you can select which of the global elements in the schema is/are to be the root elements. See SPS Structure | Schema Sources for an explanation of the possibilities offered by a selection of multiple root elements.

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

Global elements and global types

Global elements and global types can be used to create global templates which can be re-used in other templates. Additionally, global types can also be used directly in templates.

Design Fragments

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

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

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

User-defined XPath Functions

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

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

 © 2012 Altova GmbH

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

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

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

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

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

Element.

Attribute.

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

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

Design Tree

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

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

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

Toolbar icons

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

Adds a Design Fragment, main template, or layout item to the design. Clicking the left-hand part of the icon adds a Design Fragment. Clicking the dropdown arrow drops down a list with commands to add a Design Fragment or any of various layout items.

Remove the selected item; icon is active when item in the Global Templates or Layout sub-trees is selected.

Synchronize tree toggle. When toggled on (icon has border), selecting a node in the tree selects (i) the corresponding node in the design, and (ii) the corresponding node in the schema tree if the Synchronize Tree icon in the schema tree is toggled on. When toggled off, the corresponding nodes in the design and schema tree are not selected.

Auto-collapses other items in the design tree when the Synchronize Tree toggle is turned on and an item is selected in the design. Note that this toggle is enabled only when the Synchronize Tree toggle is turned on.

 © 2012 Altova GmbH

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 in the design and adds an item for it in the Design Tree. **Expand All** expands all the items of the Design Tree.

Scripts and Main Template

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

Global Templates

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

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

Design Fragments

The Design Fragments item lists all the Design Fragments in the current SPS and in all added SPS modules. Design Fragments defined in the current SPS are displayed in black, while Design Fragments that have been defined in added modules are displayed in gray (*see screenshot below*). Each Design Fragment has a check box to its left, which enables you to activate or deactivate it. A Design Fragment in the current SPS (not one in an added module)

can be removed by selecting it and clicking the **Remove** command in the context menu. The component is removed from the design and the tree.

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

XSLT Templates

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

There are two types of imported XSLT templates: (i) match templates (indicated by *Match*), and (ii) named templates (indicated by *Name*). In the Design Tree, these two types are listed with (i) the value of the `select` attribute of match templates, and (ii) by the value of the `name` attribute of named templates, respectively. For a complete description of how XSLT Templates work, see XSLT Templates.

 © 2012 Altova GmbH

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. Note that an external CSS stylesheet can also be added or a stylesheet removed via the Design Overview sidebar.

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

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

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

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

Reload All: The **Reload All** icon reloads all the external CSS stylesheets.

Reset: The **Reset** icon 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.

 © 2012 Altova GmbH

- 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.

Styles

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

The Styles sidebar is divided into two broad parts:

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

 © 2012 Altova GmbH

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.

Properties

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

The Properties sidebar is divided into two broad parts:

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

Settings for Definitions-View

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

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

Reset toolbar icon

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

Components and their property groups

The availability of property groups is context-sensitive. What property groups are available

 © 2012 Altova GmbH

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 `Value Formatting` property specifies the formatting of an Auto-Calculation that is a numeric or date datatype. The `XPath` property specifies the XPath expression that is used for the Auto-Calculation.
Common	The *Common* property group is available for all component types except the Template and AutoCalc component types. It contains the following properties that can be defined for the component: `class` (a class name), `dir` (the writing direction), `id` (a unique ID), `lang` (the language), and `title` (a name).
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 to be defined for the following client-side HTML events: `onclick`, `ondblclick`, `onkeydown`, `onkeypressed`, `onkeyup`, `onmousedown`, `onmousemove`, `onmouseout`, `onmouseover`, `onmouseup`.
HTML	Available for the following component types: data-entry devices; image; link; table; paragraphs. Note that there are different types of data-entry devices and paragraphs, and that tables have sub-components. These properties are HTML properties that can be set on the corresponding HTML elements (`img`, `table`, `p`, `div`, etc). The available properties therefore vary according to the component selected. Values for these properties can be selected using XPath expressions.

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

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

- Entered directly in the Value column. To do this, select a property, double-click in its Value column, enter the value using the keyboard, and press **Enter** or click anywhere in the GUI.
- By selecting a value from the dropdown list of the combo box for that property. Click the down arrow of the combo box to drop down the list of property-value options.
- By using the Edit button ... at the right-hand side of the Value column for that property. Clicking the Edit button pops up a dialog relevant to that property.

For some properties, in the Common and HTML groups of properties, XPath expressions can be used to provide the values of the property. The XPath icon 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

 © 2012 Altova GmbH

the XPath icon is toggled off, a static value can be entered as the value of the property. *Also see Style Properties Via XPath*.

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

Chapter 6

Quick Start Tutorial

6 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, `C:/Documents and Settings/<username>/My Documents/Altova/StyleVision2012/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.

 © 2012 Altova GmbH

6.1 Creating and Setting Up a New SPS

In this section, you will learn:

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

Creating a new SPS document

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

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

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

In a form-based document, a single Layout Container is created, in which design components can be positioned absolutely. The dimensions of the Layout Container are user-defined, and Layout Boxes can be positioned absolutely within the Layout Container and document content can be placed within individual Layout Boxes. If you wish the design of your SPS to replicate a

specific form-based design, you can use an image of the original form as a blueprint image. The blueprint image can then be included as the background image of the Layout Container. The blueprint image is used to help you design your form; it will not be included in the output.

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

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

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

Adding a schema source

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

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

2. In the Open dialog that pops up browse for the file in the (My) Documents folder, `C:/Documents and Settings/<username>/My Documents/Altova/StyleVision2012/StyleVisionExamples\Tutorial\QuickStart\QuickStart.xsd`, and click **Open**.
3. You will be prompted to select a Working XML File. Select the option to select the file from the filesystem, then browse for the file (in the (My) Documents folder) `C:/Documents and Settings/<username>/My Documents/Altova/StyleVision2012/StyleVisionExamples\Tutorial\QuickStart\QuickStart.xml`, 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.

 © 2012 Altova GmbH

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

Selecting the XSLT version

For this SPS you will use XSLT 2.0. To specify the XSLT version, in the application toolbar, click the XSLT 20 icon.

Assigning or changing the Working XML File

While adding the XML Schema to the SPS in the previous step, you also assigned a Working XML File to the schema. A Working XML File provides the SPS with a source of XML data to process. To assign, change, or unassign a Working XML File for a given schema, in the Design Overview sidebar, right-click anywhere in the Working XML File line you wish to modify (or click the Context Menu icon at the right), and select the required command from the context menu that pops up. The Working XML File is now assigned, and the filename is entered in the Design Overview. Before proceeding, ensure that you have correctly assigned the file `C:/Documents and Settings/<username>/My Documents/Altova/StyleVision2012/StyleVisionExamples\Tutorial\QuickStart\QuickStart.xml`, which is in the (My) Documents folder, as the Working XML File.

Specifying the encoding of output

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

Saving the SPS document

After you have set up the SPS as described above, save it as `MyQuickStart.sps` in the(My) Documents folder `C:/Documents and Settings/<username>/My Documents/Altova/StyleVision2012/StyleVisionExamples\Tutorial\QuickStart` folder. Do this by clicking the menu command **File | Save Design** or **Ctrl+S**, and then entering the file name in the Save Design dialog that pops up.

6.2 Inserting Dynamic Content (from XML Source)

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

Inserting element contents
In your SPS, do the following:

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

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, 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.

 © 2012 Altova GmbH

Click the HTML tab to see a preview of the HTML output (*screenshot below*). The HTML preview shows the contents of the `headline` child elements of `newsitem`, each as a text string.

Note: You can also create the contents of a node by: (i) clicking the the Insert Contents icon in the Insert Design Elements toolbar, (ii) clicking at location in the design, (iii) selecting, from the Schema Selector tree that pops up, the node for 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*).

 © 2012 Altova GmbH

4. Switch to HTML Preview 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 `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 and drop it (create it as contents) in between the start and end tags of the `dateline` element.
6. Select the `city` attribute of the `dateline/place` element (*screenshot below*) in the Schema Tree sidebar.

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

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

The HTML Preview will look like this:

 © 2012 Altova GmbH

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

Adding more dynamic content

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

Notice that the `synopsis` element has been placed before the `source` element, which is not the order in which the elements are in the schema. After you have added the `synopsis` and `source` elements to the design, check the HTML preview to see the output. This is an important point to note: that the order in which nodes are placed in the main template is how you specify the structure of the output.

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

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

6.3 Inserting Static Content

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

- An image
- A horizontal line
- Text

Inserting a static image

The static image to insert is in the (My) Documents folder: `C:/Documents and Settings/<username>/My Documents/Altova/StyleVision2012/StyleVisionExamples\Tutorial\QuickStart\NewsItems.BMP`, which 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**. The Insert Image dialog pops up (*screenshot below*).

3. In the Static tab, click the Absolute Path, then browse for the file `NewsItems.BMP` and select it.
4. Click **OK** to finish.

 © 2012 Altova GmbH

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**. A horizontal line is inserted.

Set properties for the line as follows:

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

Now insert a horizontal line at the end of each news item. To do this the cursor would have to be placed immediately before the end-tag of the `newsitem` element. This will cause the line to be output at the end of each `newsitem` element.

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

The SPS you have designed up to this point will produce output which looks something like this:

Notice that in the output of the `dateline` element, the contents of the `date` element and `place/@city` and `place/@country` attributes are run together without spacing. You can add the spacing as static text. In the design, place the cursor after the `date` element and enter a colon and a space. Next, enter a comma and space after the `@city` attribute (*screenshot below*)

This part of the output will now look like this:

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

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

Formatting static text

To format static text, highlight the text to be formatted and specify local style properties. In the design, highlight the text "`Source:  `" that you just typed. In the Styles sidebar (*screenshot below*

), notice that the *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:

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.

© 2012 Altova GmbH

6.4 Formatting the Content

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

- CSS style rules can be defined for both block components and inline components.
- Predefined formats are block components that have inherent styles and can be used as wrappers for a group of adjacent components that need to be treated as a block. The inherent styles of these predefined formats can be overridden by styles you specify.
- Class attributes can be declared on components in the design, and the class can be used as a selector for external or global style rules.
- You can specify styles at three levels. These are, in increasing order of priority: (i) style rules in external stylesheets, (ii) global style rules, and (iii) local style rules.

In this section, you will learn how to:

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

Assigning predefined formats

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

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

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

You have now grouped components together in different `div` blocks. Later in this section, 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.

 © 2012 Altova GmbH

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 (in Design View), do the following:

1. Select the `External` item.
2. Click the **Add** button in the toolbar. This pops up the Open dialog.
3. Browse for the file `C:/Documents and Settings/<username>/My Documents/Altova/StyleVision2012/StyleVisionExamples\Tutorial\QuickStart\QuickStart.css`, which is in the (My) Documents folder, and click **Open**.

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

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

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

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

NanoNull Inc Launches Version 2.0 of NanoPower

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

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

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

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

Defining local styles for multiple components at once

Local styles can be defined for multiple components at once. In your design, to specify that the entire text contents of a news item should have Arial as its font, click the `div` component surrounding the `newsitem` element and, in the Styles sidebar, in the Styles For column, select `1 paragraph`. Then, in the *font* group of properties, assign `Arial` as the `font-family`. This

property setting will be inherited by all five descendant predefined formats.

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

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

2006-04-01: Boston, USA

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

Defining local styles for a single component

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

Select the `div` component surrounding the `source` element. In the Styles sidebar, with the `1 paragraph` item in the Styles For column selected, set the `color` property (in the *color* group of properties) to `gray`. In the *font* group of 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 properties) to `black`. In the *font* group of properties, set `font-weight` to `normal`. The new properties are set on the `contents` placeholder node of the `source` element and override the properties defined on the `div` component (*see screenshot below*).

Completing the formatting

To complete the formatting in this section, select the `div` component on the `synopsis` element and, in the Predefined Formats combo box in the toolbar, select `p`. This gives the block the inherent styles of the 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.

© 2012 Altova GmbH

6.5 Using Auto-Calculations

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

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

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

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

5. In the schema tree, note that the context node is `newsitems`, which is highlighted. Now, in the Expression text box either type in the expression `count( newsitem)` or build the expression using the sidebars. (Double-click the `count` function to enter it, then place the cursor within the parentheses of the function and double-click the `newsitem` node in the schema tree.
6. Click **OK** to finish. The Auto-Calculation is inserted in the design at the cursor location (*screenshot below*). Format the Auto-Calculation using local styles.

Your HTML output will look like this:

Displaying the period covered by news items
The period covered by the news items 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 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.

6.6 Using Conditions

If you look at `QuickStart.xml`, you will see that each `newsitem` element has a `metainfo` child element, which in turn can contain one or more `relevance` child elements. 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 know 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 (*screenshot below*).

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

 © 2012 Altova GmbH

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

4. In the Edit Combo Box dialog (*screenshot above*), select Use XPath Expression and then Use the Same XPath Expression. 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.
5. Click **OK** to finish. The combo box is inserted and the design will look something like this:

6. Switch to HTML Preview. 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:

1. Select the contents of the `newsitem` part of the design which is to be contained inside the condition (highlighted in the screenshot below).

 © 2012 Altova GmbH

2. Select the menu command (or context menu command) **Enclose with | Condition**. This pops up the Edit XPath expression.
3. Enter the expression `metainfo/relevance=/presswatch/selection/byrelevance`. This expression evaluates to true when the value of the `metainfo/relevance` descendant of the current `newsitem` is the same as the value of the `/presswatch/selection/byrelevance` element (the user selection).
4. Click **OK**. The condition is created around the contents of the `newsitem` element (*screenshot below*).

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

Inserting the `relevance` node as a list

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

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

Now, in 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.

 © 2012 Altova GmbH

3. Select the `contents` placeholder and give it a local formatting (in the Styles sidebar) of bold (*font* group) and yellow background-color (*color* group).
4. Right-click the condition and, from the context menu, select **Copy Branch**.
5. In the Edit XPath Expression dialog that pops up, check the Otherwise check box (below the expression text box).
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: (i) 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 `content origin` property (in the *combo box* group of properties).
3. In the Edit XPath Expression dialog 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 te value of the `byrelevance` node. What you have to do is create a second branch for the condition, which will test for a value of `All`. By creating the news item template within this branch, you will be outputting the news item if the test is true. Do this as follows:

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

The second branch has as its contents the same template as the first branch. What the second branch does is output the news item template if the content of the `byrelevance` node is `All`.

After you have completed this section, save the design.

 © 2012 Altova GmbH

6.7 Using Global Templates and Rest-of-Contents

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

- The rules of the global template can be copied to the local template.
- A local template (in the main template) can pass processing of that node to the global template, after completing which processing resumes in the main template; in this case, the global template is said to be invoked or used.

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.

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

Inserting the `rest-of-contents` instruction

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

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

Insert the `rest-of-contents` instruction in the design by placing the cursor between the end-

tags of `newsitems` and `presswatch`, and selecting the menu command or context menu command **Insert | Rest of Contents**. The `rest-of-contents` placeholder is inserted (*screenshot below*).

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

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

Creating a global template for `selection`

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

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

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

 © 2012 Altova GmbH

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, with `table` selected in the Properties For column, specify table properties as shown in the screenshot below.

6. Set additional properties as required in the Properties and Styles sidebars. For example, a background color can be set for the header row by placing the cursor in the header row, and with `row` selected in the Properties 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

 © 2012 Altova GmbH

6.8 That's It!

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

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

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

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

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

Chapter 7

Usage Overview

7 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 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. Alternatively, a new SPS can be created directly with a schema via the **File | New** command.
2. Assign a Working XML File to the SPS. The Working XML File provides the XML data processed by the SPS when generating output previews. The Working XML File is assigned in the Design Overview sidebar. The Working XML File enables you to preview output in StyleVision.
3. Select the required XSLT version.
4. Select the Internet Explorer Compatibility to match the installed Internet Explorer version.
5. The SPS document is designed in Design View using the various design components available to the designer. The design process consists of creating a document structure and defining presentation properties.
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 are required, these are generated.

 © 2012 Altova GmbH

7.1 SPS and Sources

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

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

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

- Schema sources
- Working XML File

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

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

7.2 Creating the Design

In the SPS design, you specify:

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

Content for output
The content for the output can come from:

1. The XML document to which the SPS is applied. Content from the XML document is included in the SPS by dragging the required XML data node from the relevant schema tree in the Schema Tree sidebar and dropping this node at the desired place in the SPS .
2. An external XML document that is accessible to the application (that is, to StyleVision). By using the `doc()` function of XPath 2.0 in an Auto-Calculation, content from external XML document sources can be accessed. An XML document accessed via the `doc()` function in an XPath expression does not need to be referenced via the Schema Sources associations.
3. The SPS itself. Text and other content (such as images and tables) can be inserted directly in the SPS using the keyboard and other GUI features. Such input is independent of the XML document.
4. Manipulated dynamic (XML source) data, with the manipulations being achieved using XPath 1.0 and XPath 2.0 expressions. Manipulations are typically achieved with Auto-Calculations.
5. For the HTML output, JavaScript functions can be used to generate content.

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

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

 © 2012 Altova GmbH

7.3 XSLT and XPath Versions

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

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

XSLT transformations

XSLT transformations in StyleVision are used: (i) to generate output views in the interface; and (ii) to generate and save output files (HTML) from within the interface and from the command line. The XSLT engine used for transformations (Altova XSLT 1.0 Engine or Altova XSLT 2.0 Engine) 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 and XPath 2.0 for XSLT 2.0). XPath expressions are widely used in StyleVision—most commonly in features such as Auto-Calculations and Conditional Templates—and there are interface mechanisms that require, and help you build, XPath expressions. The functionality of the correct XPath version is automatically made available in the interface according to the XSLT version you select.

7.4 Internet Explorer Compatibility

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

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

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

- Additional image formats supported: TIFF, JPEG XR, and SVG. (SVG documents must be in XML format and must be in the SVG namespace.) These image formats will be displayed in IE9, but not in older versions of IE. For a complete listing of images supported in the various outputs, see Image Types and Output.
- Support for new CSS styles (including CSS 3 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.

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

- All CSS styles that can be rendered by the selected IE version will be automatically displayed in the Styles sidebars of StyleVision. (Note, however, that if IE 9 is selected, then IE 9 must be installed for teh IE 9-supported CSS styles to be available in the design interface.) For example, if IE 9 is installed and IE 9 is selected as the compatibility version, then the CSS 3 styles supported in IE 9 will be available in the design interface.
- HTML elements corresponding to the selected IE version can be entered as predefined formats or as user-defined elements. The HTML element will be rendered in HTML Preview according to how the installed IE version renders this element. For example, if

 © 2012 Altova GmbH

IE 9 is installed and IE 9 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 for details.

Compatibility of older SPS designs with IE 9
If you open an SPS design that has been created for an older IE version, and if the newer IE 9 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 IE 9-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 IE 9. If you change the IE compatibility to IE 9-compatibility, then check whether the table columns are generated as required. If not, you can modify the properties of the table columns or switch, in the Properties dialog, the IE compatibility back to that of the previously selected IE version.

7.5 Generated Files

In StyleVision, XSLT stylesheets and output files can be generated using the **File | Save Generated Files** command or the command line utility, `StyleVisionBatch.exe`. Alternatively, if you wish only to validate or transform XML using XSLT, you can do this directly with the Altova engines and without having to call StyleVision. The Altova engines are available at the Altova website as the free Altova product, AltovaXML.

The following files can be generated from StyleVision:

- XSLT stylesheets based on the SPS design.
- Output files generated by processing the Working XML File assigned in the SPS with the XSLT stylesheets generated from the SPS. The command line utility offers the option of specifying XML files other than the Working XML File as the XML input.

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.

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

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

- ***HTML output and stylesheets:*** (1) The formatting and layout of the generated HTML document will be identical to the HTML Preview of StyleVision. (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.

 © 2012 Altova GmbH

7.6 Catalogs in StyleVision

StyleVision supports a subset of the OASIS XML catalogs mechanism. The catalog mechanism enables StyleVision to retrieve commonly used schemas (as well as stylesheets and other files) from local user folders. This increases the overall processing speed, enables users to work offline (that is, not connected to a network), and improves the portability of documents (because URIs would then need to be changed only in the catalog files.)

The catalog mechanism in StyleVision works as outlined below.

RootCatalog.xml

When StyleVision starts, it loads a file called `RootCatalog.xml` (*structure shown in listing below*), which contains a list of catalog files that will be looked up. You can modify this file and enter as many catalog files to look up as you like, each in a `nextCatalog` element. Each of these catalog files is looked up and the URIs in them are resolved according to the mappings specified in them.

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog"
         xmlns:spy="http://www.altova.com/catalog_ext"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="urn:oasis:names:tc:entity:xmlns:xml:catalog
Catalog.xsd">
  <nextCatalog catalog="%PersonalFolder%/Altova/%AppAndVersionName%/
CustomCatalog.xml"/>
  <nextCatalog catalog="CoreCatalog.xml"/>
  <!-- Include all catalogs under common schemas folder on the first directory
level -->
  <nextCatalog spy:recurseFrom="%AltovaCommonFolder%/Schemas" catalog="
catalog.xml" spy:depth="1"/>
  <!-- Include all catalogs under common XBRL folder on the first directory
level -->
  <nextCatalog spy:recurseFrom="%AltovaCommonFolder%/XBRL" catalog="
catalog.xml" spy:depth="1"/>
</catalog>
```

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

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

CoreCatalog.xml, CustomCatalog.xml, and Catalog.xml

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

- `CoreCatalog.xml` contains certain Altova-specific mappings for locating schemas in the Altova Common Folder.
- `CustomCatalog.xml` is a skeleton file in which you can create your own mappings. You can add mappings to `CustomCatalog.xml` for any schema you require but that is not addressed by the catalog files in the Altova Common Folder. Do this using the supported elements of the OASIS catalog mechanism (*see below*).
- There are a number of `Catalog.xml` files in the Altova Common Folder. Each is inside

the folder of a specific schema or XBRL taxonomy in the Altova Common Folder, and each maps public and/or system identifiers to URIs that point to locally saved copies of the respective schemas.

Location of catalog files and schemas

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

Shell environment variables and Altova variables

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

`%AltovaCommonFolder%`	C:\Program Files\Altova\Common2012
`%DesktopFolder%`	Full path to the Desktop folder for the current user.
`%ProgramMenuFolder%`	Full path to the Program Menu folder for the current user.
`%StartMenuFolder%`	Full path to Start Menu folder for the current user.
`%StartUpFolder%`	Full path to Start Up folder for the current user.
`%TemplateFolder%`	Full path to the Template folder for the current user.
`%AdminToolsFolder%`	Full path to the file system directory that stores administrative tools for the current user.
`%AppDataFolder%`	Full path to the Application Data folder for the current user.
`%CommonAppDataFolder%`	Full path to the file directory containing application data for all users.
`%FavoritesFolder%`	Full path of the Favorites folder for the current user.
`%PersonalFolder%`	Full path to the Personal folder for the current user.
`%SendToFolder%`	Full path to the SendTo folder for the current user.
`%FontsFolder%`	Full path to the System Fonts folder.
`%ProgramFilesFolder%`	Full path to the Program Files folder for the current user.
`%CommonFilesFolder%`	Full path to the Common Files folder for the current user.
`%WindowsFolder%`	Full path to the Windows folder for the current user.
`%SystemFolder%`	Full path to the System folder for the current user.
`%CommonAppDataFolder%`	Full path to the file directory containing application data for all users.

 © 2012 Altova GmbH

%LocalAppDataFolder%	Full path to the file system directory that serves as the data repository for local (non-roaming) applications.
%MyPicturesFolder%	Full path to the MyPictures folder.

How catalogs work

Catalogs are commonly used to redirect a call to a DTD to a local URI. This is achieved by mapping, in the catalog file, public or system identifiers to the required local URI. So when the DOCTYPE declaration in an XML file is read, the public or system identifier locates the required local resource via the catalog file mapping.

For popular schemas, the PUBLIC identifier is usually pre-defined, thus requiring only that the URI in the catalog file point to the correct local copy. When the XML document is parsed, the PUBLIC identifier in it is read. If this identifier is found in a catalog file, the corresponding URL in the catalog file will be looked up and the schema will be read from this location. So, for example, if the following SVG file is opened in an XML editor that can read catalogs, such as Altova XMLSpy:

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

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

This document is read and the catalog is searched for the PUBLIC identifier. Let's say the catalog file contains the following entry:

```
<catalog>
  ...
    <public publicId="-//W3C//DTD SVG 1.1//EN" uri="schemas/svg/svg11.dtd"/>
  ...
</catalog>
```

In this case, there is a match for the PUBLIC identifier, so the lookup for the SVG DTD is redirected to the URI schemas/svg/svg11.dtd (this path is relative to the catalog file), and the local file that is referenced will be used as the DTD. If there is no mapping for the Public ID in the catalog, then the URL in the XML document will be used (in the example above: http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd).

The catalog subset supported by StyleVision

When creating entries in CustomCatalog.xml (or any other catalog file that is to be read by StyleVision), use only the following elements of the OASIS catalog specification. Each of the elements below is listed with an explanation of their attribute values. For a more detailed explanation, see the XML Catalogs specification. Note that each element can take the xml:base attribute, which is used to specify the base URI of that element.

- `<public publicId="PublicID of Resource" uri="URL of local file"/>`
- `<system systemId="SystemID of Resource" uri="URL of local file"/>`
- `<uri name="filename" uri="URL of file identified by filename"/>`
- `<rewriteURI uriStartString="StartString of URI to rewrite" rewritePrefix="String to replace StartString"/>`

- `<rewriteSystem systemIdStartString="StartString of SystemID" rewritePrefix="Replacement string to locate resource locally"/>`

In cases where there is no public identifier, as with most stylesheets, the system identifier can be directly mapped to a URL via the `system` element. Also, a URI can be mapped to another URI using the `uri` element. The `rewriteURI` and `rewritsSystem` elements enable the rewriting of the starting part of a URI or system identifier, respectively. This allows the start of a filepath to be replaced and consequently enables the targeting of another directory. For more information on these elements, see the XML Catalogs specification.

More information

For more information on catalogs, see the XML Catalogs specification.

© 2012 Altova GmbH

Chapter 8

SPS File: Content

8 SPS File: Content

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

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

 © 2012 Altova GmbH

8.1 Inserting XML Content as Text

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

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

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

1. Select the Insert Contents icon.
2. Click the location in the design where you wish to insert contents. The Insert Contents Selector pops up (*screenshot below*).

3. The context of the insertion location in the design is displayed in the *XPath Context* field. Select the node for which you wish to create contents.
4. Click **OK**. The `contents` placeholder is created. If the node you selected is anything other than the context node, additional template tags with the path to the selected node will be created around the `contents` placeholder.

Outputting text content of nodes

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

In the screenshot above, the `Desc` element has been created as contents. The output will display the text content of `Desc`. If `Desc` has descendant elements, such as `Bold` and `Italic`, then the text content of the descendant elements will also be output as part of the contents of `Desc`. Note that attribute nodes of `Desc` are not considered its child nodes, and the contents of the attribute nodes will therefore not be output as part of the contents of `Desc`. Attribute nodes have to be explicitly inserted in order to be processed.

CDATA sections

 © 2012 Altova GmbH

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 when the node is inserted.
- How descendant nodes not explicitly included within a node can be included for processing. See Rest-of-Contents.

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

Inserting Content with a Predefined Format

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

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

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

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

 © 2012 Altova GmbH

Rest-of-Contents

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

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

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

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

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

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

8.2 User-Defined Templates

User-Defined Templates are templates for items generated by an XPath expression you specify. These items may be atomic values or nodes. In the screenshot below, which shows three User-Defined Templates, note the User-Defined Template icon on the left-hand side of the tags. 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. 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)

 © 2012 Altova GmbH

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 `Location` attribute with a value of `NY`. Also see the other examples above.

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	`<xsl:for-each select="Org">`
	`  <xsl:for-each select="Office">`
	`  <xsl:for-each select="Dept">`
	`  ...`
	`  </xsl:for-each>`
	`  </xsl:for-each>`
	`</xsl:for-each>`
		Each `Office` element has its own `Dept` population. So grouping and sorting can be done within each `Office`.

Yes	`<xsl:for-each select="/Org/Office/Dept">`
	`  ...`
	`</xsl:for-each>`
		The `Dept` population extends over all `Office` elements and across `Org`.

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

Inserting a User-Defined Template

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

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

Editing a Template Match

The node selection of any node template (user-defined or normal) can be changed by using an XPath expression to select the new match expression. To edit the template match of a node

template, right-click the node template, then select the **Edit Template Match** command. This pops up the Edit XPath Expression dialog, in which you enter the XPath expression to select the new node. Then click **OK**.

Adding nodes to User-Defined Templates

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

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.

 © 2012 Altova GmbH

8.3 User-Defined Elements, XML Text Blocks

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

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

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

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.

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

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

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

4. Click **OK** to insert the element in the design. The element is displayed in the design as an empty template with start and end tags (*screenshot below*).

5. You can now add content to the template as for any other template. The User-Defined Element may contain static content, dynamic content from the XML document, as well as more additional User-Defined Elements (*see screenshot below*).

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.

 © 2012 Altova GmbH

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 obect 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).

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.

8.4 Tables

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

SPS tables

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

SPS tables are entirely presentational devices and are represented using the presentational vocabulary of 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>
```

 © 2012 Altova GmbH

```
    <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>
      <name>Name</name>
      <phone>Phone</phone>
    </person>
    <person>
      <name>John Merrimack</name>
      <phone>6517890</phone>
    </person>
    <person>
      <name>Joe Concord</name>
      <phone>6402387</phone>
    </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:

- 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 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.

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.

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,

 © 2012 Altova GmbH

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.

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.

 © 2012 Altova GmbH

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 attributes/element" 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 teh other options is given below. Click **OK** when done. Note that columns are created only for child elements and attributes, but 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:

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

 © 2012 Altova GmbH

name	ipo:name (contents) ipo:name
street	ipo:street (contents) ipo:street
city	ipo:city (contents) ipo:city
state	ipo:state (contents) ipo:state
zip	ipo:zip (contents) ipo:zip

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 in global templates. The process works in the same way as for the Root Template (given 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.

 © 2012 Altova GmbH

Tables in Design View

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

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

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

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

Drag-and-drop functionality

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

Enclosing and removing templates on rows and columns

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

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

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

 © 2012 Altova GmbH

Table Formatting

Static and dynamic tables can be formatted using:

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

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

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

Note: If all table cells in a row are empty, Internet Explorer collapses the row and the row might therefore not be visible. In this case, you should use the HTML workaround of putting a non-breaking space in the appropriate cell/s.

Vertical text

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

Note the following points:

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

 © 2012 Altova GmbH

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 *(overriden by height, width in Styles sidebar if the latter exist); align*	borders and padding in Box styles; height, width in Details group (*they override height and width in Properties sidebar*); 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 (rowspans or colspans).

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

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

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

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

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

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

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

Numbering the rows of a dynamic table

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

Table headers and footers in PDF output

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

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

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

Hyphenating content of table cells

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

 © 2012 Altova GmbH

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 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 entry helper, 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	2004-07-01 - 2004-09-30	2003-12-31	2004-01-01
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).

 © 2012 Altova GmbH

CALS/HTML Tables

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

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

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

Enabling CALS/HTML table structures for output

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

```
<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).

3. A dialog appears showing the elements of the table type you selected (*screenshot below*).

4. The element names that are listed in this dialog are, by default, the element names in the selected table model. If the SPS schema contains elements with these names then they are shown in black (*as in the screenshot above*). If a listed element name is not present in the schema, that element name is listed in red. You can change a listed element names to match a schema name by double-clicking in the relevant *Element Name* field and editing the name.
5. Click **OK** to define this XML data structure as a CALS or HTML table.
6. 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.
7. After you have finished defining the XML data structures you wish to enable as CALS/HTML tables, click **OK** to finish.

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

 © 2012 Altova GmbH

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 that pops up. Each style is added as an individual CSS attribute to the particular element. Note that a style added via the `style` attribute will have higher priority than a style added as an individual CSS attribute (such as `bgcolor`). For example, in `<thead style="background-color: red" bgcolor="blue"/>` the `style="background-color: red"` attribute will have priority over the `bgcolor="blue"` attribute.

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

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

1. The parent of the table element is inserted in the design as `( contents)` . When the contents of the parent are processed, the table element will be processed. If CALS/HTML table output is enabled, then the element is output as a table. Otherwise it is output as text.
2. The table element can be dragged from the Schema Tree. When it is dropped at the desired location in the design, it can be created as a CALS/HTML table (with the **Create CALS/HTML Table** command). If the element has not been defined as a CALS/HTML

table, the Edit 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 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).

 © 2012 Altova GmbH

8.5 Lists

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

- Static lists, which are lists, the contents of which are entered directly i the SPS. The list structure is not dynamically derived from the structure of the XML document.
- Dynamic lists, which are lists that derive their structure and contents dynamically from the XML document.

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

Static Lists

A static list is one in which list item contents are entered directly in the SPS. To create a static list, do the following:

1. Place the cursor at the location in the design where you wish to create the static list and select the **Insert | Bullets and Numbering** menu command. This pops up a dialog asking whether you wish to create a static list or dynamic list (*screenshot below*).

2. Click **Static List**. This pops up the Bullets and Numbering dialog (*screenshot below*).

3. Select the desired list item marker and click **OK**. An empty list item is created.
4. Type in the text of the first list item.
5. Press **Enter** to create a new list item.

To create a nested list, place the cursor inside the list item that is to contain the nested list and click the **Insert | Bullets and Numbering** menu command. Then use the procedure described above once again.

Note: You can also create a static list by placing the cursor at the location where the list is to be created and clicking the Bulleted List or Numbered List icons in the Insert Design Elements toolbar. The first list item will be created at the cursor insertion point.

 © 2012 Altova GmbH

Changing static text to a list
There are two ways to change static text to a list:

- Highlight the text to change, click **Insert | Bullets and Numbering**, select 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.
- With the cursor placed in a text fragment, click **Insert | Bullets and Numbering**, select the desired marker type, and click **OK**. That text fragment, till the CR-LF separators on either side, is created as a list item.

Dynamic Lists

Dynamic lists display the content of a set of sibling nodes of the same name, with each node represented as a single list item in the list. The element, the instances of which are to appear as the list items of the list, is created as the list. The mechanism and usage are explained below.

General usage mechanism

- Any element can be created as a list.
- When an element is created as a list, the instances of that element are created as the items of the list. For example, if in a `department` element, there are several `person` elements (i.e. instances), and you wanted to create a list of all the persons in the department, then you must create the `person` element as the list.
- Once the list has been created for the element, you can modify the appearance or content of the list or list item by inserting additional static or dynamic content such as text, Auto-Calculations, dynamic content, etc.

Creating a dynamic list

Create a dynamic list as follows:

1. Place the cursor at the location in the design where you wish to create the dynamic list and select the **Insert | Bullets and Numbering** menu command. This pops up a dialog asking whether you wish to create a static list or dynamic list (*screenshot below*).

2. Click **Dynamic List**. This pops up the XPath Selector dialog (*screenshot below*).
3. In the XPath Selector dialog, notice that the XPath Context is the context of the insertion location, and that it cannot be changed in the dialog. Select the node that is to be created as the dynamic list. In the screenshot below, the context node is `n1:Department`, and the `n1:Person` node has been selected as the node to be created as a list. This means that the content of each `n1:Person` node will be created as an item in the list.

 © 2012 Altova GmbH

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.

© 2012 Altova GmbH

8.6 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. What type of images are supported in the various outputs are listed in the section Image Types and Output.

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.

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 placeholders.

To insert an image, click the **Insert | Image** menu command, which pops up the Insert Image dialog (*screenshot below*).

Images can be accessed in two ways:

- The image is a file, which is accessed by entering its URI in the Insert Image dialog.
- The image is encoded as Base-16 or Base-64 text in an XML file.

Inserting an image file

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

- In the Static tab, the URI is entered directly as an absolute or relative URI. For example, `nanonull.gif` (*relative URI; see section below*), and `C:/images/nanonull.gif` (*absolute URI*).
- In the Dynamic tab, as an XPath expression that selects a node containing either (i) a URI (absolute or relative), or (ii) an unparsed entity name. For example, the entry `image/@location` would select the `location` attribute of the `image` element that is the child of the context node (that is, the node within which the image is inserted). The location node in the XML document would contain the image URI. How to use unparsed entities is described in the section Unparsed Entity URIs.
- In the Static and Dynamic tab, an XPath expression in the Dynamic part can be prefixed and/or suffixed with static entries (text). For example, the static prefix could be `C:/XYZCompany/Personnel/Photos/`; the dynamic part could be `concat(First, Last)`; and the static suffix could be `.png`. This would result in an absolute URI something like: `C:/XYZCompany/Personnel/Photos/JohnDoe.png`.

Inserting an image that is encoded text

An image can be stored in an XML file as Base-16 or Base-64 encoded text. The advantage of this is that the image does not have to be accessed from a separate file (linked to it), but is present as text in the source XML file. To insert an image that is available as encoded text in the XML source, use the Inline Data tab of the Insert Image dialog (*see screenshot below*).

 © 2012 Altova GmbH

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 Editiion; (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, 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 Image Properties 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.

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 embeded. 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/<username>/My Documents/Altova/StyleVision2012/
      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: This generates an `<img>` in the generated HTML file, and interactivity will be lost.
2. External SVG inserted as an object via the User-Defined Element feature (*see screenshot below*). Be sure to insert the `type` attribute correctly: ke

 © 2012 Altova GmbH

`type="image/svg+xml"`. When inserted in this way, the SVG object is still interactive and the mouse hover-functionality will work.

3. Inline SVG inserted via a User-Defined XML Block. See screenshot below for an example of an SVG code fragment. In this case, interactivity will work. Note that the `svg` element does not need to be in the SVG namespace if the output method is HTML 4.0 or 5.0, but the namespace is required if the output method is XHTML.

Example: A Template for Images

The StyleVision package contains an SPS file that demonstrates the use of images in StyleVision. This file is in the (My) Documents folder: `C:/Documents and Settings/<username>/My Documents/Altova/StyleVision2012/StyleVisionExamples\Tutorial\Images\Images.sps`. The Images document (`Images.xml` and `Images.sps`) consists of three parts:

- 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.

 © 2012 Altova GmbH

8.7 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.

- Input field (text box)
- Multiline input field
- Combo box
- Check box
- Radio button
- Button

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.

© 2012 Altova GmbH

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.

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).

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.

 © 2012 Altova GmbH

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.

 © 2012 Altova GmbH

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.
- You can modify the HTML properties of a combo box by selecting it and then modifying its HTML properties in the Properties sidebar.

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.

© 2012 Altova GmbH

8.8 Links

Links (or hyperlinks) can be created to bookmarks located in the document as well as to external resources like Web pages. Links can also be created to dynamically generated anchors. StyleVision offers considerable flexibility in the way target URIs for hyperlinks can be built.

The section, Bookmarks and Hyperlinks, describes how to create static and dynamic bookmarks in the document and how to link to bookmarks as well as to external documents.

8.9 Layout Modules

Layout Modules are objects containing a layout. The module as a whole is inserted in the SPS design and occurs as a block within the document flow. Within a Layout Module, multiple Layout Boxes, each containing standard SPS design elements, can be placed according to design requirements. Using Layout Modules, therefore, designers can create a layout just as they would using an artboard-based graphical design application.

The steps for creating a Layout Module are as follows:

1. Insert a Layout Container. The Layout Container can occupy the entire width of a page or can have any other dimensions you want. It can contain a blueprint of the design to serve as design guide and it can be formatted (in the Styles sidebar) using styles for the Layout Container.
2. Insert one or more Layout Boxes in the Layout Container. Layout Boxes can contain multiple design elements (including static text, schema nodes, Auto-Calculations, images, lists, etc), and they can be formatted (in the Styles sidebar) using styles for the Layout Box. Layout Boxes can also be moved relative to each other within the Layout Container and can be positioned in front of or behind each other.
3. Lines can be drawn, formatted, positioned and moved to the front or back of the stack of layout objects (Layout Boxes and other Lines).

Form-based designs
When you create a new SPS you are offered the choice of creating a free-flowing design or a form-based design. A form-based design is essentially an SPS design consisting of a Layout Container.

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

 © 2012 Altova GmbH

Layout Containers

A Layout Container has the following properties:

- It can be inserted within the flow of a document, that is, within a template. Or it can be inserted as the container within which the document design is created.
- It can have the same dimensions as the page dimensions defined for that section (the Auto-Fit to Page property of Layout Containers). Or it can have any other dimensions you specify. See the Layout Container size section below for details.
- A layout grid and a zoom feature make the positioning of objects in the Layout Container easier.
- It can have style properties, such as borders, background colors, font-properties for the whole container, etc.
- It can contain Layout Boxes and Lines, but no other design element. (All design elements must be placed within Layout Boxes.)
- It can contain a blueprint, which is an image placed on the artboard to serve as a reference template for the designer. The design can then be built to match the blueprint exactly.

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

Inserting a Layout Container

To insert a Layout Container, place the cursor at the location where the Layout Container is to be inserted and click the **Insert Layout Container** icon in the Insert Design Elements toolbar. 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.

 © 2012 Altova GmbH

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.

Note: If design element markup tags—such as template node tags—are inserted in a Layout Box, the spacing in the layout will be affected, because the tags occupy space in the layout. To avoid this source of incongruence and to match the design to the blueprint more closely, use the Hide Design Markups feature to hide tags.

 © 2012 Altova GmbH

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
- Selecting and moving Layout Boxes
- Modifying the size of the Layout Box
- Defining Layout Box style properties
- Inserting content in the Layout Box
- Stacking order of Layout Boxes

Inserting a Layout Box

A Layout Box can be inserted only in a Layout Container. To add a Layout Box, first click the Insert Layout Box icon in the Insert Design Elements toolbar, then click on the location inside the Layout Container where you wish to insert the Layout Box. A Layout Box will be inserted, with its top left corner positioned at the point where you clicked. The box will be transparent, will have no borders, and will have default text.

Selecting and moving a Layout Box

To select a Layout Box, place the cursor over the left border or top border of the Layout Box so that the cursor becomes a crossed double arrow. When this happens, click to select the Layout Box. If you keep the mouse button depressed, you can move the Layout Box to another location within its Layout Container. You can also move a Layout Box left, right, up, or down by selecting it, and then pressing the cursor key for the required direction. When the Layout Box is selected, its properties and styles are displayed in the respective sidebars.

Layout Box size

Each Layout Box has a property called *Auto-Resize* (*see screenshot below*). When the value of this property is set to `yes`, the Layout Box automatically resizes to exactly accommodate any content (including markup) that is inserted in it. 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. 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.

 © 2012 Altova GmbH

Inserting content in a Layout Box
Any type of design element can be inserted in a Layout Box, and is inserted just as it normally would be in an SPS. Note, however, that neither a Layout Container nor a Layout Line can be inserted in a Layout Box. The following points should be noted:

- When design elements are inserted that require a context node, the current node will be taken as the context node. The current node is the node within which the Layout Module has been created.
- If markup tags are displayed in a Layout Box, they would affect the WYSIWYG nature of the layout.
- 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 be displayed in the design.
- This property can also be applied to text in table cells.

Stacking order of Layout Boxes
Layout Boxes can be placed one over the other. When one Layout Box is placed on top of another, then, if it is opaque, it hides that part of the Layout Box which it covers. This behaviour 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 can also be added to a stack of Layout Boxes, and each Line can be moved relative to other items in the stack.

 © 2012 Altova GmbH

Lines

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

Inserting a Line

To add a Line to a Layout Container, do the following:

1. Click the Insert Line icon in the Insert Design Elements toolbar.
2. Click on the location inside the Layout Container where you wish to locate the start point of the line.
3. Without releasing the mouse button, draw the line from the start point to the desired end point. Then release the mouse button.

A black line will be inserted, with a dot at each end indicating the start and end points respectively.

Selecting, moving, and sizing a Line

In the Main Window, you can carry out the following-drag-and-drop functions:

- To select a Line, click any part of the Line (the cursor becomes a crossed double arrow when it is over the Line). Once a Line is selected, its properties are displayed in the Properties sidebar and can be edited there (*see below*).
- To move a Line, select it and drag it to the desired location. You can also move a line left, right, up, or down by selecting it, and then pressing the cursor key for the required direction.
- To graphically re-size or re-orient a Line, select either the start point or end point and re-position it to obtain a new size and/or orientation. You can also the 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.

 © 2012 Altova GmbH

8.10 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 comannd by right-clicking your selection. In the context menu that pops up, select **Change To** (*screenshot below*).

Changing template matches

If a template is selected, you can change the node for which that template applies. This is useful if, for instance, the name of an element has been changed in the schema. When you mouse over the Change To command and select Template from the sub-menu that pops up, you are presented with a list off all the nodes that may be inserted as a child of the selected node's parent element. Click one of these nodes to make the template apply to that node.

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

You can also change the template-match to match, not a node, but a variable template.

Changing the content type of the node

If a template or its contents are selected, then you can change the type of content the node is created as. On hovering over the Change To command in the context menu, the type of content that the selected node can be changed to is displayed as options in the sub-menu that pops up (*screenshot below*).

 © 2012 Altova GmbH

The screenshot above has been take with a combo box selected.

Chapter 9

SPS File: Structure

9 SPS File: Structure

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

- Schema sources
- Modular SPSs
- Templates and Design Fragments

Input-driven structure: schemas and modular SPS files
By input-driven, we mean that the source schemas of SPS files specify the structure of the input document/s and that this structure is the structure on which the SPS document is based. For example, if a source schema specifies a structure that is a sequence of `Office` elements, then SPS design could have a template for the `Office` element. At processing time this template will be applied in turn to each `Office` element in the source data document.

Another example of how the source document structure drives the design of the SPS file can be seen in the use of tables. Say that an `Office` element contains multiple `Person` element children, and that each `Person` element contains a set of child elements such as `Name`, `Address`, `Telephone`, etc. Then a template in the form of a table can be created for the `Person` element. Each `Person` element can be presented in a separate row of the table (*screenshot below*), in which the columns are the details of the `Person` (the child elements of the `Person` element).

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.

Additionally, StyleVision allows SPSs to be re-used as modules within other SPSs. In this way, modules can be included within a structure and can modify it. However, a schema structure contained in a module must fit in with the structure of the underlying schema of the containing SPS. How to work with modular SPSs is described in the section, Modular SPSs.

Output-driven structure: templates and design fragments
While the schema sources provide the structure of the input data document, the actual design of the output document is what is specified in the SPS document. This design is contained in one document template called the main template. The main template typically contains several component templates and can reference global templates. Templates are described in the section, Templates and Design Fragments.

This composability (of multiple templates) is further enhanced by a StyleVision feature called Design Fragments, which enables specific processing to be assigned to a document fragment that can be re-used. A Design Fragment is different than a global template in that: (i) it can be

 © 2012 Altova GmbH

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.

9.1 Schema Sources

The schema sources are the starting point of the design, and design structure can be influenced by: (i) choices you make during schema selection, and (ii) the root elements you select in the schema.

Schema selection
The selection of the schema for a new SPS file can be done in the following ways:

1. Click **File | New** and directly select a schema source to add via one of the methods (except **New (empty)**) available in the menu that pops up.
2. Click **File | New**, select New (empty) from the menu that pops up. After the new SPS is created and displayed in the GUI, in the Design Overview sidebar, select the **Add New Schema** command. This pops up a 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, then multiple root elements (document elements) can be selected for use in the design. This enables the SPS design to have templates that match multiple document elements. The advantage of this is that if an SPS, say `UniversalSPS.sps`, based on `UniversalSchema.xsd` has one template each for its two root elements, `Element-A` and `Element-B`, then this one SPS can be used with an XML instance document which has `Element-A` as its document element as well as with another XML instance document which has `Element-B` as its document element. For each XML instance, the relevant template is used, while the other is not used. This is because for the document element of each XML instance document, there is only one template in the SPS which matches that document element. For example, the document element `/Element-A` will be matched by the template which selects `/Element-A` but not by that which selects `/Element-B`. In this connection, it is important to remember that if multiple global elements are defined in the schema, an XML document with any one of these global elements as its document element is valid (assuming of course that its substructure is valid according to the schema).

To set up the SPS to use multiple root elements (document elements), click the ... button to the right of the `/Root elements` entry of the schema. The following dialog pops up.

 © 2012 Altova GmbH

The dialog lists all the global elements in the schema. Select the global elements that should be available as root elements (document elements), and click **OK**. The schema tree would then look something like this:

For the SPS represented in the screenshot above, two templates, to match both the document elements, can now be created in the design. The SPS can then be used with different XML instances that are valid according to `NanonullOrg.xsd`, some with `Department` as document element, others with `OrgChart` as document element. The appropriate template will be used in the case of each instance document.

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 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).

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 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.

 © 2012 Altova GmbH

Selecting files via URLs
To select a file via a URL, do the following:

1. Click the **Switch to URL** command. This switches to the URL mode of the Open dialog (*screenshot below*).

2. Enter the URL you want to access, in the *Server URL* field (*screenshot above*). If the server is a Microsoft® SharePoint® Server, check the *Microsoft® SharePoint® Server* check box. See the Microsoft® SharePoint® Server Notes below for further information about working with files on this type of server.
3. If the server is password protected, enter your User-ID and password in the *User* and *Password* fields.
4. Click **Browse** to view and navigate the directory structure of the server.
5. In the folder tree, browse for the file you want to load and click it.

The file URL appears in the File URL field (*screenshot above*). The **Open** button only becomes active at this point.

6. Click the **Open** button to load the file. The file you open appears in the main window.

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

Note: To give you more control over the loading process, you can choose to load the file through the local cache or a proxy server (which considerably speeds up the process if the file has been loaded before). Alternatively, you may want to reload the file if you are working, say, with an electronic publishing or database system; select the **Reload** option in this case

Microsoft® SharePoint® Server Notes

Note the following points about files on Microsoft® SharePoint® Servers:

- In the directory structure that appears in the Available Files pane (*screenshot below*), file icons have symbols that indicate the check-in/check-out status of files.

 © 2012 Altova GmbH

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

- The various file icons are shown below:

	Checked in. Available for check-out.
	Checked out by another user. Not available for check-out.
	Checked out locally. Can be edited and checked-in.

- After you check out a file, you can edit it in your Altova application and save it using **File | Save (Ctrl+S)**.
- You can check-in the edited file via the context menu in the Open URL dialog (*see screenshot above*), or via the context menu that pops up when you click the file tab in the Main Window of your application (*screenshot below*).

- When a file is checked out by another user, it is not available for check out.
- When a file is checked out locally by you, you can undo the check-out with the Undo Check-Out command in the context menu. This has the effect of returning the file unchanged to the server.
- If you check out a file in one Altova application, you cannot check it out in another Altova application. The file is considered to be already checked out to you. The available commands at this point in any Altova application supporting Microsoft® SharePoint® Server will be: **Check In** and **Undo Check Out**.

Opening and saving files via Global Resources

To open or save a file via a global resources, click **Switch to Global Resource**. This pops up a dialog in which you can select the global resource. These dialogs are described in the section, Using Global Resources. For a general description of Global Resources, see the Global Resources section in this documentation.

The `anyType` datatype of XML Schema

If an element in the XML Schema has been assigned the `anyType` datatype of XML Schema or if it has not been assigned any datatype, then the schema tree in the Schema Tree will show this element as having all the global elements of that schema as possible children. For example, if an element called `email` has not been assigned any datatype, then it will be displayed in the schema tree with all global elements as possible children, such as, for example: `person`, `address`, `city`, `tel`, etc. To avoid this, assign the `email` element a datatype such as `xs:string`.

 © 2012 Altova GmbH

User-Defined Schemas

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

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

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

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

2. In the Root Elements tree, there is a single root element (document element) called `UserRoot`.
3. Double-click `UserRoot` and rename it to match the document element of the XML document for which you are building this schema.
4. To assign a child element or an attribute to the document element, select the document element (`UserRoot`), and click, respectively, the Append New Element icon or the Append New Attribute icon in the toolbar of the Schema Tree sidebar. Alternatively, you can right-click and select the required command from the context menu. After the new element or attribute is added to the tree, type in the desired name. Note that the Append New Element icon and append New Attribute icon have dropdown menus that can be accessed by clicking the dropdown arrow in the respective icon. The dropdown menus contain items that enable you to add nodes at alternative levels relative to the selected node. You can also drag nodes to the desired location (described in the next step). In the screenshot below, the `Article` element is the document element. The elements `Title`, `Para`, `Bold`, and `Italic`, and the attributes `ID` and `Author` have been added at the child level of `Article`.

5. To move the elements `Bold` and `Italic`, and the attribute `ID` to the level of children of `Para`, select each individually and drag to the `Para` element. When a bent downward-pointing arrow appears, drop the dragged node. It will be created as a "child" of `Para` (*screenshot below*).

6. When any element other than the document element is selected, adding a new element or attribute adds the new node at the same level as the selected element. Drag a node (element or attribute) into an element node to create it as a "child" of the element node.

 © 2012 Altova GmbH

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.

9.2 Merging XML Data from Multiple Sources

XML data from multiple source XML files can be merged when XSLT 2.0 is used as the XSLT version of the SPS.

Typically, the merging of data will be based on a common piece of data, such as an ID. For example, an employee in a company, who is identified by a personal ID number, can have his or her personal data stored in multiple XML files held by the personnel department: (i) personal details, (ii) payroll, (iii) work and leave, (iv) courses attended, etc. Data from these different files can be merged in a single output document using the personal ID number as a key.

Note: The Enterprise Edition enables you to include multiple schema sources, so XML nodes from other schemas can be selected using the parameter name for the corresponding schema (as is the case in the example below). In the Professional and Basic Editions, the `doc()` function of XPath 2.0 can be used to locate the required XML file and the XML node within that file. The `doc()` function of XPath 2.0 provides access to the document root of external XML documents, and thus enables node content from external XML documents to be inserted in the output. An Auto-Calculation that uses the `doc()` function can, therefore, also be used to merge XML data (*see example below*).

Example

The (My) Documents folder, `C:/Documents and Settings/<username>/My Documents/Altova/StyleVision2012/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>
```

 © 2012 Altova GmbH

```
   <Article PartNum="238-KK">
      <ProductName>Amber ring</ProductName>
      <Price>89.90</Price>
   </Article>
   <Article PartNum="229-OB">
      <ProductName>Pearl necklace</ProductName>
      <Price>4879.00</Price>
   </Article>
   <Article PartNum="128-UL">
      <ProductName>Jade earring</ProductName>
      <Price>179.90</Price>
</Article>
      ...
</Articles>
```

The way the merging of the data is done is to set up a User-defined template within the `/Order/Item` template (*see screenshot below*) that selects the corresponding `Article` element in the `MergeArticles.xml` file by using the part number of the ordered item to identify the article. The XPath expression (which is in the `/Order/Item` context) is: `$Articles//Article[@PartNum=current()/@partNum]`

This template will produce output something like that shown in the screenshot below.

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` template is selected within the context of the `/Articles/Article` template.

The same result as that obtained above could also be achieved using an Auto-Calculation (*see screenshot below*).

The XPath expression of the Auto-Calculation could target the required node using either the parameter of another schema source or the `doc()` function:

```
$Articles//Article[@PartNum=current()/@partNum]/ProductName
```

or

```
doc('MergeArticles.xml')//Article[@PartNum=current()/@partNum]/ProductName
```

Notice that, while the first XPath expression above uses a parameter to refer to another XML Schema (a feature available only in the Enterprise Edition), the second expression uses the `doc()` function of XPath 2.0 (a feature available in the Professional and Basic editions as well).

 © 2012 Altova GmbH

9.3 Modular SPSs

The global templates of an SPS, as well as Design Fragments, JavaScript functions, and page layout items can be used in the design of another SPS. This enables:

1. The re-use of global templates and other components across multiple SPSs, the main advantages of which are single-source editing and consistency of output.
2. SPSs to be modularized, and thus to be more flexibly structured.

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

Available module objects
The section, Available Module Objects, not only describes the extent to which, and conditions under which, the various components of an SPS are available to an importing SPS. It also lists those components that are not available to the importing SPS. You should note that if an added module itself contains modules, then these are added recursively to the referring SPS. In this way, modularization can be extended to several levels and across a broad design structure.

Creating a modular SPS
To build a modularized SPS, first add the required SPS to the main SPS as a module. All the global templates, Design Fragments, JavaScript functions, and page layout items 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 and page layout items 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. And JavaScript functions are assigned via the Property window as event handlers for the selected design component.

How to create and work with a modular SPS is described in the section, Creating a Modular

SPS.

Terminology
When an SPS is used within another module it is said to be added to the latter, and we call the process **adding**. The two SPSs are referred to, respectively, as the ***added SPS module*** and the ***referring SPS module***. When an SPS module is added, its **objects** are added to the referring SPS module. These objects are called ***module objects***, and are of the following types: global templates; Design Fragments; JavaScript functions; and page layout items.

 © 2012 Altova GmbH

Available Module Objects

This section lists the objects in added SPS modules that are available to the referring SPS module. The listing explains in what way each object is available to the referring SPS module and how it can be used there. For a step-by-step approach to creating modular SPSs, see the next section, Creating a Modular SPS. This section ends with a list of objects in the added SPS that are not available to the referring SPS module, which will help you to better understand how modular SPSs work.

- Namespace declarations
- Global templates
- Design fragments
- Added modules
- Scripts
- CSS styles
- Page layouts
- Unavailable module objects

Namespace declarations

Each SPS stores a list of namespace URIs and their prefixes. When an SPS is added as a module, the namespaces in it are compared to the namespaces in the schema source/s of the referring SPS. If a namespaces 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 referring SPS. If a namespace in the added SPS cannot be matched with any 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 of the added SPS module are available to the referring SPS module and are displayed in the Design Tree sidebar (*screenshot below*).

Note that the main template of added modules are not available. This means that if you plan to re-use a template via the modular approach, you must create it as a global template. If no global template is defined for a particular element and processing is invoked for that element, then the default processing for that element (XSLT's built-in templates) will be used.

Design fragments
Design fragments in the added SPS module are available to the referring SPS and are displayed in the Design Tree sidebar (*screenshot above*). When inserting a design fragment in the design, care should be taken to place the design fragment within the correct context node in the design.

Added modules
Each added SPS module also makes available to the referring SPS its own added modules, and their added modules, and so on. In this way, adding one module recursively makes available all modules that have been added to it, down multiple levels. Needless to say, these modules must together construct a content model that is valid according to the source schema/s of the referring SPS module. Modules are displayed and can be managed in the Design Overview sidebar.

Scripts
The scripts in all the added SPS modules are available for use in the referring SPS and are displayed in the Design Tree sidebar. In effect, the scripts of all the added modules are collected in a library that is now—in the referring SPS—available for selection in the Properties dialog.

CSS styles
The global styles present in added SPS modules are carried over to the referring SPS as global

 © 2012 Altova GmbH

styles and the style rules are displayed in the Style Repository sidebar. The CSS files are also listed in the Design Overview sidebar. Similarly, external CSS files that were available to the added SPS module, are available to the referring SPS module.

Page layouts
The page layouts of an added module are available to the referring SPS and are displayed in the Design Tree sidebar.

Module objects that are not available to the referring SPS
The following objects of the added module are not available to the referring SPS:

- ***Parameter definitions:*** are ignored.
- ***Schema sources:*** The schema source on which the added SPS is based is ignored. Bear in mind that the content model of the document element of the added SPS must be contained within the content model of the referring SPS; otherwise it would not be possible to correctly use the added SPS as a module. If you wish, you could always add a 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.

Creating a Modular SPS

Creating a modular SPS consists of three broad parts:

1. Design and save the SPS module to be added.
2. Add the module to the SPS in which it is to be used (that is, to the referring SPS module).
3. Activate or deactivate the added object/s as required.
4. Apply the required object whereever required.

The SPS module to be added

There are two points to bear in mind when creating an SPS that will be added to another:

1. The templates that can be used in the referring SPS module can only be global templates. This means that the templates you wish to re-use must be created as global templates in the SPS module that is to be added.
2. The document structure defined in the SPS module to be added must be valid within the content model defined by the source schema/s of the referring SPS. If an added template is not contained in the content model defined by the main schema of the SPS, its content model, however, can still be defined in a 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 used since these determine the expanded names of nodes.

You could use a Working XML File to test the output of the SPS module to be added. The reference to this Working XML File will be ignored by the referring SPS.

Adding the SPS module

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

 © 2012 Altova GmbH

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, select it in the Design Overview and click, as required, the **Move Up** or **Move Down** toolbar icon in the Design Tree toolbar.

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`; and `Page Layout`. 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.

 © 2012 Altova GmbH

Applying or using modular objects
In the referring SPS module, you design your templates as usual. Each different type of added object is used or applied differently. You should, of course, ensure that each module object you wish to apply has been activated.

Global templates
When you wish to use a global template from any of the added SPS modules, you must make sure that this global template is indeed applied. This can be done in one of two ways, according to which one is appropriate for your design:

- In the main template, specify that the element template either uses the global template for that element or copies that global template locally. These two commands are available in the context menu that appears when you right-click the element tag in the design.
- In the main template, the contents or rest-of-contents placeholders cause templates to be applied, leading to the relevant global templates being processed.

Design Fragments
To use a Design Fragment, drag it from the Design Tree to the desired location in the main template or a global template. Make sure that the location where the Design Fragment is dropped is the correct context node for that Design Fragment. For details, see Design Fragments.

Scripts
All JavaScript functions (whether in an added module or created in the referring SPS) are available as event handlers, and can be set for a particular event via the Properties sidebar.

Page layout items
If page layout items have been defined in any of the added modules, these are listed under the Page Layout item of the referring SPS module. If a page layout item is not required, it can be unchecked. Where there is more than one option for the same item, for example, `HeaderOdd`, then you can select which one of the options is to be applied by checking that option's check box.

Example: An Address Book

The (My) Documents folder, `C:/Documents and Settings/<username>/My Documents/Altova/StyleVision2012/StyleVisionExamples\Tutorial\ModularSPS`, contains examples of modular SPSs. The example files in this folder comprise a project in which an address book containing business and personal contacts is modularized. The example not only demonstrates the mechanisms in which modularization is implemented, but also illustrates the main reasons why one would modularize.

- The complete address book is composed of two modules: (i) a business address book, and (ii) a personal address book, each of which has a separate SPS defining different designs. The two modules together make up the composite address book. Modularization in this case is used to compose: the modules are the components of a larger unit.
- Although the content model of each module (business and personal address books) differs slightly from the other, both have a common module, which is the ContactPoints module, consisting of the core contact details: address, telephone, fax, and email. The ContactPoints module can therefore be shared between the two address books (business and personal). Modularization in this case enables a single module to be used as a common unit within multiple other units.
- Further, the ContactPoints module can be modularized to provide more flexibility. In the example project, we have created a separate Address module to contain the postal address, which may have one of three content models, depending on whether the address is in the EU, US, or elsewhere. The output for all three content models is defined in a single SPS. However, they could have been defined in separate SPSs, which would have provided finer granularity. In this case, modularization would provide more flexibility as modules could be re-used more easily.

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

- The schema files
- The XML data sources
- The SPS files

The schema files

When creating schemas for modular SPSs, the most important thing to bear in mind is to create the elements that you wish to re-use as global elements. The schema for the address book is `AddressBook.xsd`. This schema has been constructed by importing the schemas for the business address book (`BusinessAddressBook.xsd`) and personal address book (`PersonalAddressBook.xsd`). The `BusinessAddressBook.xsd` schema provides a content model for companies, while the `PersonalAddressBook.xsd` schema provides a content model for persons (*see screenshot below*).

Both schemas import the `ContactPoints.xsd` schema (*see screenshot below*), which defines a content model for contact details.

Finally, the `ContactPoints.xsd` schema (*screenshot below*) includes the `Address.xsd` schema, which defines the three address-type content models: for EU, US, and other addresses.

Imports are used when the imported schema belongs to a different namespace than the importing schema. Includes are used when the included schema belongs to the same namespace as the including schema.

Note: The screenshots above are of the schema in the Schema/WSDL View of Altova's XMLSpy.

The XML data sources

The XML data is contained in the file `AddressBook.xml`. This file is structured so that the

AddressBook element contains the companies and persons elements as its children. The content models of these two elements are defined in the schema files, BusinessAddressBook.xsd and PersonalAddressBook.xsd, respectively.

There are two additional XML data files, which correspond to the BusinessAddressBook.xsd and PersonalAddressBook.xsd schemas. These two XML files, BusinessAddressBook.xml and PersonalAddressBook.xml, are used as the Working XML Files of the corresponding SPS files.

The three XML files are the Working XML Files of the following SPS modules:

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

The SPS modules

The description of the SPS modules starts with the most basic module (Address.sps) and progresses in compositionally incremental steps to the complete address book (AddressBook.sps). All the SPS modules use AddressBook.xsd as its schema.

Address.sps

The key points to note are the use of the schema and the Working XML File.

- Address.sps uses AddressBook.xsd as its schema, but the schema could equally well have been Address.xsd, ContactPoints.xsd, BusinessAddressBook.xsd, or PersonalAddressBook.xsd—since the Address element is present in all these schemas and would be available as a global element. When the SPS module is added to another SPS module, the schema of the imported module is ignored, so which one is used is not important when the SPS is added as a module.
- The Working XML File is AddressBook.xml. Note that the main template in Address.sps specifies that only the Address element should be processed, and that global templates for Address-EU, Address-US, and Address-Other have been defined.

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*).

 © 2012 Altova GmbH

- 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
This SPS imports 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.

© 2012 Altova GmbH

9.4 Templates and Design Fragments

The design document is composed of templates, and it is important to recognize the various types of templates that can be used.

- *Main templates and global templates:* The design document consists of one main template and, optionally, one or more global templates. Global templates can be referenced via the main template.
- *Node-templates and variable iterators:* These are the templates that constitute the main template and global templates. A node-template matches a node in a schema source.
- *Design fragments:* These are templates that are designed separately and re-used in various parts of the design (main template or global templates).

In this section, we describe the role that templates and design fragments play in the structure of the design. We are not concerned here with the presentation properties in the design, only the structure.

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, which are available as toolbar icons. These display filters will help you optimize and switch between different displays of your SPS.

Main Template

The main template determines the structure of the output. That is, 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. After a local template is processed, the processor moves to the next component in the main template, and so on. Occasionally, a node may reference a global template for its processing. In such cases, after the global template is executed for that node, the processor returns to the position in the main template from which it branched out and continues in sequence from the next component onwards.

The entry point for the main template is the document node of the schema. StyleVision offers the option of selecting multiple root elements (document elements). This means that within the main template, there can be local templates for each of the active document elements. The one that is executed during processing will be that for the element which is the document element of the XML instance document being processed.

 © 2012 Altova GmbH

Global Templates

A global template can be defined for any node or type in the schema, or for a node specified in an XPath pattern.

A global template specifies instructions for the selected node or type, and it is invoked by a call from the main template, design fragments, or other global templates. The processing model is similar to that of declarative programming languages, in that a single template is defined and invoked multiple times. In this way a single definition can be re-used multiple times. Global templates are invoked in two situations:

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

Global templates are useful if a node (or type) occurs within various elements or in various locations, and a single set of instructions is required for all occurrences. For example, assume that a `para` element must be formatted the same no matter whether it occurs in a `chapter`, `section`, `appendix`, or `blockquote` element. An effective approach would be to define a global template for `para` and then ensure, that in the main template the global template for the `para` element is processed wherever required (for example, by including `//chapter/para` in the main template and specifying that `para` reference its global template; or by including `//chapter/title` and then including `(contents)` or `(rest-of-contents)` so that the rest of the content of the `chapter` element is processed with the available global templates and default templates). Also, a global template can be defined for a complex type (for example, one that defines an address model) or even for a simple type (for example, `xs:decimal`). In such cases, all occurrences of the type (complex or simple) that invoke the global template for that type will be processed according to the rules in the global template.

Creating a global template

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

A global template can be created in any of the following ways:

- Click the **Add New Global Template** button located at the right of the Global Templates item in the Schema Tree (*see screenshot above*). This pops up the Add New Global Template dialog (*screenshot below*). You can select an element, an attribute, or a type from the schema tree shown in the dialog, or you can enter an XPath pattern. 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 **Add New Global Template**. This pops up the Add New Global Template dialog, which is described above.
- 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*).

Using a global template

After a global template has been created, it can be used when a node having the same qualified

 © 2012 Altova GmbH

name is inserted into the document (by dropping . Alternatively, if a local template is present in the design and a global template exists for a node having the same qualified name, then the global template can be used instead of the local template. To use a global template for a local template, right-click the local template in Design View and select the command **Use Global Template**. When a global template is used, its processing instructions are called and used by the local template at runtime.

Wherever a global template is used in the design, an XPath pattern can be created on the global template to filter the nodeset it addresses. To create such a filter, right-click the global template tag in the design, and select **Edit XPath Filter** in the context menu that appears. This pops up the Edit XPath Expression dialog, in which the required expression can be entered.

Recursive global templates

Global templates can be recursive, that is, a global template can call itself. However, to guard against an endless loop in Authentic View, a property to limit the call-depth can be set. This property, the *Maximum Call-Depth* property, is available in the Authentic tab of the Properties dialog of the SPS (**File | Properties**). It specifies the maximum number of template calls that may be made recursively when processing for the Authentic View output. If the number of template calls exceeds the number specified in the *Maximum Call-Depth* property, an error is returned.

Copying a global template locally

After a global template has been created, its processing instructions can be copied directly to a template of the same qualified name in the main template. To do this, right-click the local template and select the command **Copy Global Template Locally**. Copying the global template locally is different than using the global template (at runtime) in that the processing instructions are merely copied in a one-time action. The global template has no further influence on the local template. Either, or both, the global template and local template can subsequently be modified independently of each other, without affecting the other. On the other hand, if it is specified that a global template should be *used* (at runtime) by a local template, then any modifications to the global template will be reflected in the local template at runtime.

Activating and deactivating global templates

A global template can be activated by checking its entry in the global templates listing in the Schema Tree sidebar. It can be deactivated by unchecking the entry. If a global template has been activated (the default setting when the global template was created), it is generated in the XSLT stylesheet. If it has been deactivated, it is not generated in the XSLT stylesheet but is still saved in the SPS design.

Any local template that uses a deactivated global template will then—since it is not able to reference the missing global template—fall back on the default templates of XSLT, which have the collective effect of outputting the contents of descendant text nodes.

The advantages of the activation/deactivation feature are: (i) Global templates do not have to be deleted if they are temporarily not required; they can be reactivated later when they are required; (ii) If there are name conflicts with templates from imported stylesheets, then the global template that is not required can be temporarily deactivated.

Removing a global template

To remove a global template, right-click the global template to be removed, either in Design View or the Schema Tree sidebar, and select the command **Make/Remove Global Template**.

Simple global templates and complex global templates
Global templates are of two types: simple and complex. Complex global templates are available for reasons of backward-compatibility. If a global template in an SPS created with a version of StyleVision prior to version 2006 contains a table or list, then that global template will typically be opened in StyleVision 2006 and later versions as a complex global template.

A complex global template is different than a simple global template in the way the node for which the global template was created is processed. When the first instance of the node is encountered in the document, the complex global template processes all subsequent instances of that node immediately afterwards. A simple global template, on the other hand, processes each node instance only when that node instance is individually encountered.

It is important to note that a simple global template will be automatically converted to a complex global template if a predefined format or newline is created **around** the element node for which the global template was created. This will result in the processing behaviour for complex global templates (described in the previous list item). To revert to the simple global template, the predefined format should be removed (by dragging the node outside the predefined format and then deleting the predefined format), or the newline should be removed (by deleting the item in the Design Tree sidebar), as the case may be. To avoid the automatic conversion from simple global template to complex global template, make sure that the predefined format or newline is added within the node tags of the element for which the simple global template was created.

Global templates in modular SPSs
When an SPS module is added to another SPS module, the global templates in the added module are available for use within the referring SPS. For more information about using modular SPSs, see the section Modular SPSs.

 © 2012 Altova GmbH

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. 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. 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 `Location` attribute with a value of `NY`. Also see the other examples above.

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	`<xsl:for-each select="Org">`
	`  <xsl:for-each select="Office">`
	`  <xsl:for-each select="Dept">`
	`  ...`
	`  </xsl:for-each>`
	`  </xsl:for-each>`
	`</xsl:for-each>`
		Each `Office` element has its own `Dept` population. So grouping and sorting can be done within each `Office`.

Yes	`<xsl:for-each`
	`select="/Org/Office/Dept">`
	`  ...`
	`</xsl:for-each>`
		The `Dept` population extends over all `Office` elements and across `Org`.

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

Inserting a User-Defined Template

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

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

Editing a Template Match

The node selection of any node template (user-defined or normal) can be changed by using an XPath expression to select the new match expression. To edit the template match of a node template, right-click the node template, then select the **Edit Template Match** command. This

 © 2012 Altova GmbH

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.

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.

To insert a variable template, do the following:

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

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

 © 2012 Altova GmbH

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 screenshot below contains two node-templates: `metainfo` and `relevance`.

The operations that can be carried out on a node-template are accessible via the context menu of that node-template (accessed by right-clicking either the start or end tag of a node-template, *see screenshot below*).

The commands in this context menu are described below:

- Global templates
- Template match
- XPath filters
- Group by, Sort by, Define variables, Template serves as level
- Create Design Fragment
- Remove Tag Only

- Edit, Enclose with, Change to, Authentic properties

These menu commands are described below. Note that for a given node-template, some commands might not be available; these are grayed out in the context menu.

Global templates: make, use, copy locally
A node-template in the main template can be changed to or associated with a global template via the following commands:

- *Make global template:* This option is available if the node-template represents an element that is defined as a global element in the schema. A global template will be created from the node-template. The node-template in the main template will use this global template and its tags will then be displayed in gray (indicating its use of the global template).
- *Use global template:* If a global template of the same qualified name as the node-template has been defined, the node-template will use the processing of the global template. The tags of the node-template will become gray.
- *Copy global template locally:* The processing instructions of a global template of the same qualified name as the node-template are copied physically to the node-template. The node-template is independent of the global template. Subsequently, both it and the global template can be modified independently of each other. Since the node-template does not reference a global template, it retains its beige color.

For more information, see the section Global Templates.

Editing the template match
The node for which a template has been created can be changed by using this command. The Edit Template Match command pops up the Edit XPath Expression dialog, in which you can enter an XPath expression that selects another node in the schema. You can also enter any XPath expression to change the template to a User-Defined Template.

Edit/Clear XPath Filter
An XPath filter enables you to filter the nodeset on which a node-template is applied. XPath filters can also be applied to global templates.

By default, a node-template will be applied to nodes (elements or attributes) corresponding to the node for which the node-template was created (having the same name and occurring at that point in the schema hierarchy). For example, a node-template for the `/Personnel/Office` node will select all the `/Personnel/Office` elements. If an XPath filter with the expression `1` is now created on the `Office` element (by right-clicking the `Office` element and editing its XPath Filter), this has the effect of adding a predicate expression to the `Office` element, so that the entire XPath expression would be: `/Personnel/Office[1]`. This XPath expression selects the first `Office` child of the `Personnel` element, effectively filtering out the other `Office` elements.

A filter can be added to any node-template and to multiple node-templates in the design. This enables you to have selections corresponding to such XPath expressions as: `/Personnel/Office[@country='US']/Person[Title='Manager']` to select all managers in the US offices of the company. In this example, a filter each has been created on the `Office` and on the `Person` node-templates, respectively.

Wherever a global template is used—that is, called—an XPath filter can be applied to it. So, for

 © 2012 Altova GmbH

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.
- The **Sort by** command enables instances of the node represented by the selected node-template to be sorted. The sorting mechanism is described in the section, Sorting.
- The **Define Variables** command enables you to define variables that are on scope on the selected node-template. How to work with variables is described in the section, Variables.
- The **Template Serves as Level** command is a toggle command that creates/removes a level on the node-template. Levels can be specified at various levels in order to structure the document into a hierarchy. This structure can then be used to generate a table of contents (TOC), automatic numbering, and text references. These features are described in detail in the section, Table of Contents (TOC) and Referencing.

Create Design Fragment
Creates a Design Fragment template from the selected template. The resulting Design Fragment template is added to the Design Fragment templates at the bottom of the design, and added to the Design Tree and Schema Tree. The Design Fragment is also applied at that point in the design where it was created.

Remove (Template or Formatting) Tag Only
This command removes the selected template or formatting tag only. It does not remove any descendant nodes or formatting tags. This command is useful for removing a formatting tag or a parent element tag without removing all that is contained within the tag (which is what would happen if the **Delete** operation is carried out with a tag selected). Note, however, that removing a parent element might render descendant nodes of the deleted element invalid. In such cases, the invalid nodes are indicated with a red strike-through.

Edit, Enclose with, Change to, Edit Authentic Properties
These commands are described below:

- *Edit:* Pops out a submenu with the familiar Windows commands: cut, copy, paste, and delete.
- *Enclose with:* The node-template can be enclosed within the following design components, each of which is described in a separate section of this documentation: paragraph, special paragraph, Bullets and Numbering, Hyperlink, Condition, TOC Bookmark and Level.
- *Change to:* The Change-To feature enables you to change: (i) the node for which that template applies, or (ii) how the node is created in the design. It is described in detail in the section, The Change-To Feature.

 © 2012 Altova GmbH

Design Fragments

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

1. Create the Design Fragment in the design
2. Fill out the contents of the Design Fragment
3. Insert the Design Fragment at a location in a template.

Creating a Design Fragment

To create a Design Fragment do the following:

1. In the Design Tree or Schema Tree, click the Add New Design Fragment icon, which is located to the right of the Design Fragments item in the tree (*see screenshot below*). This adds a Design Fragment item in the Design Fragments list of the tree. (*Also see note below.*)

Notice that a Design Fragment template is created in the SPS design. This template is appended to the templates already in the design. (If you wish to see only the Design Fragments that are in the design, hide the main template and global templates by clicking their Show/Hide icons in StyleVision's Design Filter toolbar.) Additionally, the Design Fragment templates are also listed in the schema tree for ready access from there.

2. Double-click the Design Fragment item (either in the design tree or the schema tree) so as to edit its name. Name the Design Fragment as required and press **Enter**. The edited name is entered in the Design Tree (*screenshot below*) and in the template in the design.

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.

Creating the contents of a Design Fragment
The contents of the Design Fragment template are created as for any other template. To insert static content, place the cursor in the Design Fragment template and insert the required static content. To insert dynamic content, drag the required schema node into the Design Fragment template.

When dragging a node from the schema source you can drag the node either: (i) from the Global Elements tree, or (ii) from the Root Elements tree. The difference is significant. If a node is dragged from the Global Elements tree, it is created without its ancestor elements (in the screenshot below, see the `EmailPerson` Design Fragment) and, therefore, when used in a template, it will have to be used within the context of its parent. On the other hand, if a node is dragged from the Root Elements tree, it is created within a structure starting from the document node (in the screenshot below, see the `EmailDocNode` Design Fragment), and can therefore be used anywhere in a template.

The screenshot above shows two Design Fragment templates that produce identical output for the `Person` element. In the `EmailPerson` Design Fragment template, the `Person` node has been created by dragging the global element `Person` into the `EmailPerson` template. In the `EmailDocNode` Design Fragment template, the `Person` node has been dragged from the Root Elements tree, and is created with an absolute path (from `$XML`, the document node).

When these Design Fragment templates are inserted in the main template, care must be taken that the `EmailPerson` template is called from within a context that is the parent of the `Person` node. You can experiment with these Design Fragments. They are in the example file `Email.sps`, which is in the (My) Documents folder, `C:/Documents and Settings/<username>/My Documents/Altova/StyleVision2012/StyleVisionExamples\Tutorial\DesignFragments`.

You can also define a parameter with a default value on the Design Fragment. The parameter can be assigned a different value in every Design Fragment instance. See Parameters for Design Fragments for details.

 © 2012 Altova GmbH

After you have completed the design, notice that the components of the design are also graphically depicted in the Design Tree.

Inserting a Design Fragment in a template
To insert a Design Fragment, drag the Design Fragment from the Design Tree or Schema Tree to the required location. The location at which the Design Fragment is dropped should be such that it provides a correct context. If the contents of the Design Fragment were created from a global element, then the correct context in the main template would be the parent of the node dragged into the Design Fragment. See Creating the contents of a Design Fragment above.

Alternatively, right-click at the location where the Design Fragment is to be inserted and select **Insert Design Fragment** from the context menu.

Note: If a Design Fragment is referenced in the main template and if the name of the Design Fragment is changed subsequently, then the reference in the main template will no longer be correct and an XSLT error will result. In order to correct this, delete the original reference in the main template and create a fresh reference to the newly named Design Fragment.

Recursive design fragments
Design fragments can be recursive, that is, a design fragment can call itself. However, to guard against an endless loop in Authentic View, a property to limit the call-depth can be set. This property, the *Maximum Call-Depth* property, is available in the Authentic tab of the Properties dialog of the SPS (**File | Properties**). It specifies the maximum number of template calls that may be made recursively when processing for the Authentic View output. If the number of template calls exceeds the number specified in the *Maximum Call-Depth* property, an error is returned.

Deleting a Design Fragment
To delete a Design Fragment, select it in the Design Tree and click the **Remove** toolbar icon of the Design Tree .

Design Fragments in modular SPSs
When an SPS module is added to another SPS module, the Design Fragments in the added module are available for use within the referring SPS. For more information about using modular SPSs, see the section Modular SPSs.

Example file
For an example SPS, go to the (My) Documents folder, `C:/Documents and Settings/<username>/My Documents/Altova/StyleVision2012/StyleVisionExamples\Tutorial\DesignFragments`.

9.5 XSLT Templates

XSLT files can be imported into an SPS, and XSLT templates in them will be available to the stylesheet as global templates. If, during the processing of the XML document, one of the XML nodes matches a node in an imported XSLT template, then the imported XSLT template is applied to that node. If the imported XSLT file contains named templates, these are available for placement in the design.

Note:

- Imported XSLT templates cannot be modified in StyleVision.

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, and then the **Add XSLT File** command.

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.

 © 2012 Altova GmbH

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.

```
<xsl:call-template name="BoldItalic" />
```

The effect of this in the output is to implement the named template at that location in the design. This can be useful for inserting content that is independent of both the XML instance document as well as of the XSLT stylesheet.

9.6 Multiple Document Output

You can design an SPS to produce multiple output-documents: a main output-document and one or more additional documents. This is particularly useful if you wish to modularize the output. Output-documents are created in the design by inserting a New Document template (*see screenshot below*). Content for each output-document is placed within its New Document template.

New Document templates can be created anywhere in the document design, thus allowing the output to be modularized at any level. So, for example, a report about the various branch offices of a global organization can have separate output-documents at each of the following levels: (i) world, (ii) continent, (iii) country, (iv) state, and/or (v) branch office. Each branch office, for example, can be presented in a separate output-document or all the branch offices in a country can appear together in a single country report. In the design, a New Document template would have to be created at each of the hierarchical levels for which separate output-documents are required. How to set up the correct document structure is described in the section, New Document Templates and Design Structure.

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

- Inserting a New Document Template
- New Document Templates and Design Structure
- URLs of New Document Templates
- Preview and Output Document Files
- Document Properties and Styles

 © 2012 Altova GmbH

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** within a New Document template. The New Document template will, in this case, contain the enclosed content when it is created. You can add to or modify this content in the design. To place a New Document template so that it encloses content, highlight the content to be enclosed and then select the command **Enclose With | New Document**. Alternatively, you can select the content to be enclosed, then right-click it, and, from the context menu that pops up, select the command **Enclose With | New Document**.

A New Document template with content is shown in the screenshot below.

Notice the following from the screenshot above:

1. The New Document template tags contain the URL (path and name) of the output-document it will generate. The filename suffix will be generated automatically according to the file type of the output format. For example, for the HTML output format, the filename suffix `.html` will be appended to the filename in the URL. Issues relevant to the assigning of URLs are discussed in the section, URLs of New Document Templates .
2. The New Document Template contains one Initial Document Section. Additional document sections can be added to the initial document section as described in the section, Document Sections.

New Document Templates and Design Structure

When creating multiple output-documents, you must create the different New Document templates on the appropriate nodes of the source document. Therefore, you must consider both the output structure as well as the input (source XML document) structure when designing multiple output-documents.

Main output document and additional output documents (output structure)
When the first New Document template is added to the design, all design content outside this New Document template is automatically assigned to a separate document. This separate document is considered to be the main output document, and, in the output previews of StyleVision, it is referred to as Main Output Document.

In the generated output-documents (created using the command **File | Save Generated Files**), the name of the main output document will be the name you assign it when generating the output-document files using the **Save Generated Files** command. The names of the additional output-document files will be the names assigned in the URLs of the respective New Document templates.

New Document templates and source document structure
When a New Document template is created, the hierarchical location where it is created is significant. Two possibilities exist:

1. The node within which the New Document template is created is processed only once. In this case the New Document template is also processed only once. The filename in the URL property of the New Document template can therefore be a static name.
2. The node within which the New Document template is created is processed multiple times. As a result, the New Document template will be processed as many times as the node is processed. An example of such a situation would be the following. An `Office` element has multiple `Department` element children (for its various departments). If a New Document template is created within the `Department` node in the design, then, since the `Department` node will be processed multiple times (for all the different `Department` elements in that `Office` element), the New Document template also will be processed multiple times, once for each `Department` element in the source XML document. The filename in the URL property of the New Document template must therefore be a dynamic name. Otherwise, the output-documents created for the Department elements will each have the same filename.

 © 2012 Altova GmbH

URLs of New Document Templates

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

URLs of New Document templates

If the New Document template is processed only once (*see preceding section*), then the template's *URL* property can be a static URL. In the screenshot below, since the New Document template is immediately within the document element (`$XML`), it will be processed only once. The URL has been given a static value of `TableOfContents`. This value will therefore be the filename of the output-document. Since no path has been prefixed to the filename, the file will be generated in the same directory as the Main Document File (*see* Multiple Document Outputs and Previews for details). Alternatively, if the URL contained a path, the output-document will be saved to the location specified in the path.

If, on the other hand, a New Document template will be processed multiple times to generate multiple output-documents (*see preceding section*), then the template's *URL* property must be a dynamic URL that is selected with an XPath expression. In the screenshot below, the URL of the New Document template is the XPath expression: `body/header/para`. The New Document template is within the `topic` element, so it will be processed each time the `topic` element is processed. On every iteration through the `topic` element, the content of that `topic` element's `body/header/para` element will be assigned as the URL of the New Document template. This creates a new document for every `topic` element. Each of these documents has a different name, that of its `body/header/para` element (which is the text of the topic's header).

Editing the URL
When a New Document template is added to the design, it is created with a default URL. This is a static text string: `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 sidebar (*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 sidebar, 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. A bookmark can be placed at the head of a New Document Template or anywhere within the New

 © 2012 Altova GmbH

Document Template. Hyperlinks can then be created in other documents to link back to the bookmark. If bookmarks are required on a node that is processed multiple times, then make sure that the name of the bookmark is generated dynamically. Otherwise (if a static bookmark name is given) multiple nodes in the output will have the same bookmark name.

A Table of Contents (TOC) can also be used to link documents. The TOC could be in a separate document (for example, the main document) and link to the various output-documents, while the output-documents could link back to the TOC.

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.

 © 2012 Altova GmbH

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.

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.

 © 2012 Altova GmbH

Chapter 10

SPS File: Advanced Features

10 SPS File: Advanced Features

How to create the basic content and structure of the SPS design is described in the sections, SPS File Content and SPS File Structure. Very often, however, you will also need to modify or manipulate the content and/or structure of source data in particular ways. For example, you might wish to sort a group of nodes, say nodes containing personnel information, on a particular criterion, say the alphabetical order of employee last names. Or you might wish to group all customers in a database by city. Or add up a product's sales turnover in a particular city. Such functionality is provided in StyleVision's advanced features, and these are described in this section.

Given below is a list of StyleVision's SPS file advanced features:

- Auto-Calculations. Auto-Calculations are a powerful XPath-based mechanism to manipulate data and (i) present the manipulated data in the output as well as (ii) update nodes in the XML document with the result of the Auto-Calculation.
- Conditions. Processing of templates and the content of templates can be conditional upon data structures or values in the XML, or upon the result of an XPath expression
- Grouping. Processing can be defined for a group of elements that are selected with an XPath expression.
- Sorting. A set of XML elements can be sorted on multiple sort-keys. .
- Parameters and Variables. Parameters are declared at the global SPS level with a default value. These values can then be overridden at runtime by values passed to the stylesheet from the command line. Variables can be defined in the SPS and these variables can be referenced for use in the SPS.
- Table of Contents (TOC) and Referencing. Tables of Contents (TOCs) can be constructed at various locations in the document output, for all output formats. The TOC mechanism works by first selecting the items to be referenced in the TOC and then referencing these marked items in the TOC. Other features which use referencing are: (i) Auto-Numbering (repeating nodes in the document can be numbered automatically and the numbers formatted; (ii) Text References (text in the document can be marked for referencing and then referenced from elsewhere in the document); and (iii) Bookmarks and Hyperlinks (bookmarks mark key points in the output document, which can then be targeted by hyperlinks. Hyperlinks can also link to external resources using a variety of methods to determine the target URI (static, dynamic, a combination of both, and unparsed entity URIs).) All these referencing mechanisms are described in this section.

 © 2012 Altova GmbH

10.1 Auto-Calculations

The **Auto-Calculation** feature (i) displays the result of an XPath evaluation at any desired location in the output document, and (ii) optionally updates a node in the main XML document (the XML document being edited in Authentic View) with the result of the XPath evaluation.

The Auto-Calculation feature is a useful mechanism for:

- Inserting calculations involving operations on dynamic data values. For example, you can count the number of `Employee` elements in an `Office` element (with `count( Employee)`), or sum the values of all `Price` elements in each `Invoice` element (with `sum( Price)`), or join the `FirstName` and `LastName` elements of a `Person` element (with `concat( FirstName, ' ', LastName)`). In this way you can generate new data from dynamically changing data in the XML document, and send the generated data to the output.
- Displaying information derived from the structure of the document. For example, you can use the `position( )` function of XPath to dynamically insert row numbers in a dynamic table, or to dynamically number the sections of a document. This has the advantage of automatically generating information based on dynamically changing document structures.
- Inserting data from external XML documents. The `doc( )` function of XPath 2.0 provides access to the document root of external XML documents, and thus enables node content from the external XML document to be inserted in the output.
- Updating the value of nodes in the main XML document. For example, the node `Addressee` could be updated with an XPath expression like `concat( Title, ' ', FirstName, ' ', LastName).`
- Presenting the contents of a node at any location in the design.

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 sidebar 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. (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, then you must use XPath 2.0 expressions.)

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 (*screenshot above*), in which you can edit the XPath expression.

 © 2012 Altova GmbH

Formatting Auto-Calculations
You can apply predefined formats and CSS styles to Auto-Calculations just as you would to normal text: select the Auto-Calculation and apply the formatting. Additionally, input formatting of an Auto-Calculation that is a numeric or date datatype can be specified via the Input Formatting property in the AutoCalc group of properties in the Properties window.

Note also that you can include carriage returns and/or linefeeds (CR/LFs) in the XPath expression. If the Auto-Calculation is enclosed in the `pre` special paragraph type, the output of a CR/LF will produce a new line in the output. 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.

Example: An Invoice

The `SimpleInvoice.sps` example in the (My) Documents folder, `C:/Documents and Settings/<username>/My Documents/Altova/StyleVision2012/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*).

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

 © 2012 Altova GmbH

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-amount = Net Price * VAT-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. 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.

10.2 Conditions

You can insert conditions anywhere in the design, in both the main template and global templates. A condition is an SPS component that is made up of one or more branches, with each branch being defined by an XPath expression. For example, consider a condition composed of two branches. The XPath expression of the first branch tests whether the value of the `Location` attribute of the context node is "`US`". The XPath expression of the second branch tests whether the value of the `Location` attribute is "`EU`". Each branch contains a template—a condition template. When a node is processed with a condition, the first branch with a test that evaluates to true is executed, that is, its condition template is processed, and the condition is exited; no further branches of that condition are evaluated. In this way, you can use different templates depending on the value of a node. In the example just cited, different templates could be used for US and EU locations.

This section consists of the following topics:

- Setting Up the Conditions, which describes how to create a condition and its branches.
- Editing Conditions, about how to edit the XPath expressions of condition branches after they have been created.
- Conditions and Auto-Calculations, explains usage issues when conditions and Auto-Calculations are used in combination.

 © 2012 Altova GmbH

Setting Up the Conditions

Setting up the condition consists of the following steps:

1. Create the condition with its first branch.
2. Create additional branches for alternative processing.
3. Create and edit the templates within the various branches of the condition.

Creating the condition with its first branch

Set up a condition as follows:

1. Place the cursor anywhere in the design or select a component and then select the menu command **Insert | Condition**.The Edit XPath Expression dialog pops up (*screenshot below*).

2. In the Expression pane, enter the XPath expression for the condition branch via the keyboard. Alternatively, enter the expression by double-clicking nodes, operators, and/or functions in the sidebar panes of the dialog. It is important to be aware of the context node at the insertion point; the context node is highlighted in the schema sources tree when the dialog pops up.
3. Click **OK** to finish. The condition is created with its first branch; the XPath expression you entered is the XPath expression of the first branch. If the condition was inserted at a text insertion point, the first branch is empty (there is no template within it; *see screenshot below*). If the condition was inserted with a component selected, the condition is created around the component, and that component becomes the template of the first branch.

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

branch, click the cell with the number one.

After creating a condition with one branch (which may or may not have a template within it), you can create as many additional branches as required.

Creating additional branches
Additional branches are created one at a time. An additional branch is created via the context menu (*screenshot below*) and can be created in two ways: (i) without any template within it (**Add New Branch**); and (ii) with a copy of an existing template within the new branch (**Copy Branch**).

To create a new branch, right-click any branch of the condition and select **Add New Branch** from the context menu. The Edit XPath Expression dialog will pop up. After entering an XPath expression and clicking **OK**, a new empty branch is added to the condition. This is indicated in the design by a new cell being added to the condition; the new cell has a number incremented by one over the last branch prior to the addition.

To create a copy of an existing branch, right-click the branch of the condition you wish to copy and select **Copy Branch**. The Edit XPath Expression dialog will pop up, containing the XPath expression of the branch being copied. After modifying the XPath expression and clicking **OK**, a new branch is added to the condition. The new branch contains a copy of the template of the branch that was copied. The new branch is indicated in the design by a new cell with a number incremented by one over the last branch prior to the addition.

The Otherwise branch
The `Otherwise` branch is an alternative catch-all to specify a certain type of processing (template) in the event that none of the defined branches evaluate to true. Without the `Otherwise` branch, you would either have to create branches for all possible eventualities or be prepared for the possibility that the condition will be 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 dialog click the Otherwise check box (*screenshot below*).

 © 2012 Altova GmbH

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**.

Editing Conditions

To edit the XPath expression of a condition branch, do the following:

1. Select the condition branch (not the condition).
2. In the Properties sidebar, select `condition branch` in the Properties For column (*screenshot below*).

3. Click the **Edit** button ... of the `XPath` property in the *When* group of properties. This pops up the Edit XPath Expression dialog, in which you can edit the XPath expression for that branch of the condition.

 © 2012 Altova GmbH

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.

10.3 Grouping

The grouping functionality is available in **XSLT 2.0** 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*).

 © 2012 Altova GmbH

In the dialog, check the Enable Grouping check box, then select the required Grouping Type and, in the Match text box, enter the XPath expression that defines the grouping key (for the *group-by* and *group-adjacent* options) or the desired match pattern (for the *group-starting-with* and *group-ending-with* options). When you click **OK**, a dialog pops up asking whether you wish to sort the group-set alphabetically (in ascending order). You can always sort group-sets subsequently or remove such sorting subsequently. The screenshot below shows nodes and current-group templates which have had grouping added to them.

In the screenshot above, the `person` node has been grouped and the resulting groups sorted. For example if the `person` elements have been grouped by department, then the various departments can be sorted in alphabetically ascending order. The groups thus created have been further grouped by creating grouping on the current-group() template. In this way `person` elements can be grouped, say, first by department, and then by employment grade.

Sorting groups

After confirming a grouping definition, a pop-up asks you to confirm whether the groups should be sorted in ascending order or not. You can set sorting subsequently at any time, or modify or delete, at any time, the sorting set at this stage.

To set, modify, or delete sorting subsequently, right-click the required grouping template and select **Sort by**. This pops up the Define Output Sort Order dialog. How to use this dialog is described in the section Sorting. The important point to note is that to sort groups on the basis of their grouping-key, you must select the XPath function `current-grouping-key()` as the sorting key. For examples, see the files described in the following sections.

Viewing and editing grouping and sorting settings

To view and edit the grouping and sorting settings on a template, right-click the template and select Group by or Sort by, respectively. This pops up the respective dialog, in which the settings can be viewed or modified.

User-defined templates

User-defined templates are templates that are applied to items selected by an XPath expression you specify. The nodes selected by the XPath expression of a user-defined template can also be grouped. In this case, the grouping is applied on the user-defined template.

 © 2012 Altova GmbH

Example: Group-By (Persons.sps)

The `Persons.sps` example is based on the `Persons.xsd` schema and uses `Persons.xml` as its Working XML File. It is located in the (My) Documents folder, `C:/Documents and Settings/<username>/My Documents/Altova/StyleVision2012/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 **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 **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, to group by grade, create a grouping for the `grade` attribute on the `current-group()` template. Confirm the default sorting.

8. Sort the current group (which is the sub-group of persons sorted 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.
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.

 © 2012 Altova GmbH

Example: Group-By (Scores.sps)

The `Scores.sps` example is based on the `Scores.xsd` schema and uses `Scores.xml` as its Working XML File. It is located in the (My) Documents folder, `C:/Documents and Settings/<username>/My Documents/Altova/StyleVision2012/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

 © 2012 Altova GmbH

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 **OK**. (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 (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.

10.4 Sorting

The sorting functionality is available in XSLT 1.0 and XSLT 2.0 SPSs and 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 are templates that are applied to items selected by an XPath expression you specify. The nodes selected by the XPath expression of a user-defined template can also be sorted. In this case, the sorting is applied on the user-defined template.

In this section

- The sorting mechanism is described.
- An example demonstrates how sorting is used.

 © 2012 Altova GmbH

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.

User-defined templates
User-defined templates are templates that are applied to items selected by an XPath expression you specify. The nodes selected by the XPath expression of a user-defined template can also be sorted. In this case, the sorting is applied on the user-defined template.

A note about sort-keys
In both XSLT 1.0 and XSLT 2.0 SPSs, 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); the key for each element should be resolvable to a string or number value.

In an **XSLT 2.0** 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.

 © 2012 Altova GmbH

Example: Sorting on Multiple Sort-Keys

In the simple example below (available in the (My) Documents folder, `C:/Documents and Settings/<username>/My Documents/Altova/StyleVision2012/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 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, `members`, 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.

 © 2012 Altova GmbH

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

10.5 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 explains how user-defined parameters can be used in an SPS.
- Parameters for Design Fragments describes how parameters can be used with design fragments.
- SPS Parameters for Sources are a special type of parameter. They are automatically defined by StyleVision for schema sources (specifically, the Working XML Files of schemas). Since the name and value of such a parameter are known to the user, the parameter can be referenced within the SPS and a value passed to it at run-time from the command line.
- Variables enable you to: (i) declare a variable with a certain scope and define its value, and (ii) to reference the value of declared variables and create a template on a node or nodes selected by the variable.

 © 2012 Altova GmbH

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 the command line.

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 the command line.

Usage mechanism

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

1. Declaring the required parameters.
2. Referencing the declared parameters.

Declaring parameters

All user-defined parameters are declared and edited in the Edit Parameters dialog (*screenshot below*). The Edit Parameters dialog is accessed via: (i) the **Edit | Stylesheet Parameters** command and (iii) the Edit button ... of the Parameters entry in the Design Overview sidebar.

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. The default value will be used each time the parameter is referenced, and it is overridden only if a new value is passed for that parameter on the command line.

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.

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.

 © 2012 Altova GmbH

Parameters for Design Fragments

Parameters for Design Fragments enable you to define a parameter on a design fragment you have created and to give this parameter a default value. At each location where this design fragment is used in the design, you can enter a different parameter value, thus enabling you to modify the output of individual design fragments.

For example, a design fragment named `EMailAddresses` can be created with a parameter named `Domain` that has a default value of `altova.com`. Now, say this parameter is used in an Auto-Calculation in the design fragment to generate the email addresses of company employees. For the EU addresses, we could use the design fragment `EmailAddresses` and edit the value of the `Domain` parameter to be `altova.eu`. In the same way, in the template for Japanese employees, we could edit the value of the `Domain` parameter to be `altova.ja`. For the US employees of the company, we could leave the parameter value of `Domain` unchanged, thus generating the default value of `altova.com`.

Using parameters for design fragments consists of two parts:

1. Defining the parameter with a default value on the design fragment where it is created.
2. Editing the parameter value where the design fragment is used.

These parts are explained in detail below.

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

Defining the parameter

Each design fragment can be assigned any number of parameters. To do this, click the Define Parameters link in the title bar of the design fragment (*see screenshot below*).

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, exactly 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 `EmailAddress`, 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*).

 © 2012 Altova GmbH

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.

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.

 © 2012 Altova GmbH

Variables

Using variables consists of two parts: (i) declaring the variable, and (ii) using the variable.

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

Declaring a variable

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

Declare a variable as follows:

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

3. Setting a variable to Editable (by checking the *Editable* check box) enables the variable to be edited in Authentic View (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, 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.

 © 2012 Altova GmbH

10.6 Table of Contents, Referencing, Bookmarks

The Table of Contents (TOC) and 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. Two types of anchoring mechanism are used:

- A simple anchor is created at a point in the design document. The anchor (or bookmark) is given a unique name and this name is used as the target of links that point to this document fragment. This mechanism is used for the Bookmarks and Hyperlinks feature. Links can additionally point to URLs outside the document.
- For more complex referencing, such as for TOCs and the auto-numbering of document sections, building the anchor involves two parts. First, the design document is structured into the hierarchy required for the TOC. This is achieved by assigning levels to different points in the document structure. For example, a level can be assigned to a book chapter. Second, the text that will appear in the referencing component must be defined. After the levels and the reference text have been defined, the referencing component can be designed. This mechanism is broadly described below, under The TOC mechanism.

The various referencing features are explained in detail in the rest of this section.

The TOC mechanism

If you have selected **XSLT 2.0** (not XSLT 1.0) as the XSLT version of your SPS, you can create a table of contents (TOC) at any location in the design. The mechanism for creating the TOC consists of two parts, which are described in the sub-sections of this section:

- The items from the design that are to be included in the TOC are marked in the design. 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 marks the `header` item for inclusion in the TOC.
- A template is created for the TOC (*highlighted in screenshot below*). The TOC template contains the design of the TOC; it can be located anywhere in the design. In the example shown in the screenshot below, the TOC template is located near the top of the document.

Either of these two parts can be created first, or both parts can be created concomitantly.

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, in which items are first marked for inclusion, and the TOC template is constructed subsequently. (Alternatively, you can create the TOC template first, and then mark items for inclusion; or you can create the TOC template and select items for inclusion in parallel.)

1. Make sure that XSLT 2.0 is the selected XSLT version.

2. *Structure the document in levels*. If the TOC is to have multiple levels, structure the design in a hierarchy of nested levels. If the TOC is to have a flat structure (that is, one level only), then create at least one level that will enclose the TOC bookmarks.
3. *Create one or more TOC bookmarks within each level in the document design.* The TOC bookmarks identify the components within each level that are to appear in the TOC.
4. *Create a TOC template*. The TOC template should have the required number of TOC reference levels (reflevels). In the case of a multi-level TOC, the reflevels in the TOC template should be nested (*see screenshot above*).
5. *Create TOCrefs*. In the TOC template, set up a TOCref for each level. Each TOCref will reference, by name, the required TOC bookmarks within that level in the document; alternatively, the TOCref may additionally reference TOC bookmarks in other levels.
6. *Format the TOC items*. Each TOC item (in the TOC template) can contain item numbering (including hierarchical), the TOC item text, a leader, and, for paged media, a page number. Each TOC item and its parts can be formatted as required. Note that you can include numbering not only in the TOC template, but also within a TOC bookmark in the main body of the document.

Marking Items for TOC Inclusion

Marking 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 levels. A level is created in the design either on a template or around a design component. In the screenshot below, a level has been created on the `topic` template topic.

When a level is created on a template, this is indicated by the level icon inside the start tag of the template. For example, topic. When a level is created around a component it is indicated by level tags level level. In the screenshot above, the `topics` template component is enclosed by a level. The difference between the two ways of marking levels is explained in the section Structuring the Design in Levels. When the TOC template is created, it must be structured in a hierarchy of levels, with the levels in the TOC template corresponding to the levels you have created in the design. Even for TOCs with a flat structure (one level), the design must have a corresponding level.

2. Creating a TOC bookmark in the design with a name and TOC-item text. The TOC bookmark can either enclose or not enclose a design component; in the latter case it is empty. In the screenshot below, the TOC bookmark does not enclose a design component.

The TOC bookmark serves as an anchor in the document. In the screenshot above, the TOC bookmark (and anchor) is located at the start of `para` element instances. The TOC bookmark has two attributes: (i) a name that will be used to reference the TOC bookmark when creating the TOC item in the TOC template, and (ii) a text string that will be used as the text of the corresponding TOC item. How these two attributes are assigned is described in the section, Creating TOC Bookmarks.

How marked items are referenced in the TOC template

The TOC template is structured in nested levels (called reference levels (reflevels) to differentiate them from the levels created in the main body of the design template). Within each reflevel, a TOC reference (TOCref) 'toc' is inserted (*see screenshot below*). The TOCref within a level references TOC bookmarks using the TOC bookmark's name. Each TOC bookmark with that name and which is within the corresponding level in the XML document will

 © 2012 Altova GmbH

be created as a TOC item at this level in the TOC (when the scope of the TOCref is specified to be the current level). For example, the TOCref indicated with the tag 'chapters' 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 bookmark will be output as the text of the TOC item.

In the screenshot above of a TOC template, there are three nested reflevels, within each of which is a TOCref that contains the template for the TOC item of that level. For example, in the first level, there is a TOCref that references TOC bookmarks that have a name of `toc` 'toc' . As a result, all TOC bookmarks in the first level (as structured in the design) and named `toc` will be accessed for output at this level in the TOC. The TOCref within the second level also references TOC bookmarks having a name of `toc`. As a result, all TOC bookmarks in the second level of the document (as structured in the design) and that are named `toc` will be used for second-level items in the TOC. The third level works in the same way: TOC bookmarks named `toc` that occur within the document's third level are referenced for third-level items in the TOC.

In the sub-sections of this section, we describe: (i) how the design is structured into levels, and (ii) how bookmarks are created. How the TOC template is created is described in the section, Creating the TOC Template.

Structuring the Design in 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.

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 hierarchical structure for the output of the SPS.

Creating levels

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

To create a level, do the following:

1. Select the component (template or other).
2. Right-click, and from the context menu select **Template Serves As Level** (enabled when a template is selected) or **Enclose With | Level**. Both these options are also available in the **Insert | Insert Table of Contents** menu: **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).

 © 2012 Altova GmbH

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 for the first level of `topic` topic. The `rest-of-contents` instruction in the main template specifies that templates will be applied for all child elements of `topic/body` except `header`. This means that the global template for `topic` children of `topic/body` will be processed. In the global template for `topic`, a level has been set on the `topic` template (indicated by topic). This second level of the TOC hierarchy, which occurs on the second level of `topic` elements, is nested within the first level of the TOC hierarchy. Since this global template also has a `rest-of-contents` instruction, the global template for `topic` will be applied to all recursive `topic` elements, thus creating additional nested levels in the TOC hierarchy.

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 a flat TOC hierarchy

In a flat TOC hierarchy, the TOC items will be output at a single level; the outline of the document in the TOC will be a simple list of items. In the TOC template, the items to be listed are referenced in the usual way in the design document: by their name and the level in which they occur. Therefore, the document design must contain at least one level, and this level must contain all the required TOC bookmarks.

If the design contains more than one level, and the flat TOC is required, say, for items in the second level, then the TOC template could have two reflevels with a TOCref within the second level (*screenshot below*).

For example, consider the design document shown in the screenshot above: It has one level on the `topic` template in the main template and sub-levels on the `topic` template in the global template. The TOC template shown in the screenshot below will produce a flat TOC of the second-level topic headers (assuming that the bookmark name is `toc`).

This is because the TOCref in the TOC template references TOC bookmarks named `toc` that are within the second level. Notice that in the TOC template the TOCref item is created within the second reflevel of the TOC template. Since only one level is output (there is no output for the first reflevel), the resulting TOC will be flat.

Note: Alternatively, the scope attribute of TOCrefs can be used to specify what level/s in the design document should be looked up for bookmarks of a given name.

 © 2012 Altova GmbH

Creating TOC Bookmarks

TOC bookmarks are created within a TOC level in the document design. They can be created in the main template and/or in global templates. A TOC bookmark serves two purposes:

- It marks a (static or dynamic) component in the design with a static name you assign. It can either enclose or not enclose a design component; in the latter case it is empty. In the output, the TOC bookmark is instantiated as an anchor identified by a name.
- It defines the text string that will be used as the text for the TOC item/s. 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 the TOC bookmark in two ways: (i) by using the Create Marker Wizard, which enables you to specify the TOC bookmark's name; its text entry; whether auto-numbering should be used; and the level within which it appears; and (ii) by inserting an empty TOC bookmark, the properties of which will be defined subsequently.

Creating the TOC bookmark with the Create Marker Wizard

To create a TOC bookmark using the TOC Bookmark Wizard, 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 **Insert** menu, select **Insert Table of Contents | TOC Bookmark (Wizard)**. This pops up the Create Marker Wizard (*screenshot below*).

1. In the wizard's first screen (*screenshot above*) you: (i) define the text entry for the TOC item; (ii) set the TOC bookmark (or marker) name; and (iii) specify whether this marker should be numbered in the output. For the text entry you can select whether the text of child elements should be used, or the result of an XPath expression. For the name of the marker, you can enter text directly or select from a dropdown list containing the names of already specified marker names. When you are done, click **Next**.
2. In the wizard's second screen (*screenshot below*), you can select the level within which the TOC bookmark is to be inserted.

Ancestor templates on which levels are assigned are indicated with a level icon (in the screenshot above, the `topic` template has a level). Select a template-level within which the TOC bookmark is to be created. If a level already exists for this template, the TOC bookmark will be created within this level, otherwise a new level will be created on the selected template. Alternatively, you can choose to define the level later by checking the Define Level Later check box. When you are done, click **Finish**.

Creating a TOC bookmark

To create a TOC bookmark without attributes, 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 **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.

Inserting hierarchical or sequential numbering for a component

Hierarchical or sequential numbering can be inserted within a TOC bookmark's tags. Right-click at the location where you wish to insert the numbering, then select **Insert Table Of Contents | Hierarchical / Sequential Numbering**. Since numbering can only be inserted at locations within a TOC bookmark, it is better, for numbering purposes, that a TOC bookmark be created around a component rather than be empty. This would allow greater layout flexibility in the placement of the numbering.

Editing the name and text entry of a TOC bookmark

The name and text entry of the TOC bookmark can be edited in the Properties window (*screenshot below*). To edit these properties, select the TOC bookmark, and either directly edit the property in the Property window or right-click the TOC bookmark and select the property you wish to edit.

 © 2012 Altova GmbH

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

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 in the design document. If you select multiple levels for the TOC (next option), the same TOCref name will be used in all levels (though individual TOCref names can be edited subsequently). (ii) The number of TOC reflevels specifies how many levels 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 reflevels (*screenshot below; the formatting of the TOC template has been modified from that which is created initially*).

Within each reflevel is a TOCref having a name that identifies TOC bookmarks that are to be the TOC items for that TOC template reflevel. Within each TOCref is a default template for the TOC item, which you can edit at any time.

 © 2012 Altova GmbH

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.
- Reflevels can be added to or deleted from the structure of the TOC template.
- The properties of individual TOC references (TOCrefs) can be edited. The name and scope of a TOCref can be changed, and you can choose whether the TOC item corresponding to the TOCref is created as a hyperlink or not.
- TOCrefs can be added to or deleted from any reflevel in the TOC template.
- The TOC item within a TOCref can be formatted with CSS properties using the standard StyleVision mechanisms.
- Standard SPS features (such as images, Auto-Calculations, and block-formatting components can be inserted anywhere in the TOC template.

Reflevels in the TOC Template

The TOC template is structured in **level references (or reflevels)**; *see screenshot below*. These levels are initially created when the TOC template is created, and the number of reflevels are the number you specify in the Create TOC Page dialog.

Notice that the reflevels are nested. For the purposes of the TOC design there is a one-to-one correspondence between the reflevels in the TOC template and the levels in the SPS design. Thus, the first reflevel of the TOC template corresponds to the first level in the SPS design, the second reflevel in the TOC template to the second level in the SPS design, and so on. The TOCrefs within a given reflevel of the TOC template identify TOC bookmarks within a specified scope in the SPS design.

Inserting and removing reflevels

Reflevels can be inserted in or deleted from the TOC template after the TOC template has been created.

To insert a reflevel, select the content in the TOC template around which a reflevel is to be created, then select **Insert | Insert Table of Contents | Level Reference**. Alternatively, from the context menu, select **Enclose With | Level Reference**. A reflevel can also be inserted at a cursor insertion point in the TOC template.

To remove a reflevel from the TOC template, select the reflevel to be removed and either press the **Delete** key or select **Remove** from the context menu. Note that only the reflevel will be removed—not its contents.

 © 2012 Altova GmbH

TOC References: Name, Scope, Hyperlink

TOC references (TOCrefs) occur within level references (reflevels) and have three properties:

- A ***name***, which identifies TOC bookmarks of the same name that occur within the specified scope as the items to be included at that level of the TOC.
- A ***scope***, which specifies to which corresponding levels in the SPS design the TOCref applies. Three options are available: global, current level, current level and descendant levels.
- A ***hyperlink*** property which can be toggled between `yes` and `no` to specify whether the corresponding TOC items are created as hyperlinks or not.

To insert a TOCref, place the cursor within a reflevel 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 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).

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, 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 | Text Entry / Leader / Page Reference** or **Insert Table Of Contents | Hierarchical Numbering / Sequential Numbering** as required. 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.

Formatting the TOC item

The TOC item can be formatted with CSS styles via the Styles sidebar. Individual TOC item components can be separately formatted by selecting the component and assigning it style properties in the Styles sidebar.

 © 2012 Altova GmbH

Example: Simple TOC

An example SPS file to demonstrate the basic use of TOCs, called `ChaptersSimple.sps`, is in the (My) Documents folder, `C:/Documents and Settings/<username>/My Documents/Altova/StyleVision2012/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.)

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 two steps:

1. Structuring the design in TOC levels: One or more levels are inserted in the design document to structure the (output) document hierarchically. This hierarchic structure will be the one that the TOC reflects. In our current example a single level has been created—on the `Topic` template. As a result, there is only one level in the design document and the TOC can have only one level in its structure.
2. Creating TOC bookmarks: A TOC bookmark is created within the level that was created

in Step 1 (in the design document). This enables TOC References in the TOC template to point back to this bookmark. The TOC bookmark also contains the text that will appear in the TOC item that points to it.

3. Creating the TOC template: This is the template that creates the TOC in the document. It is structured into Level References (reflevels), which must correspond to the structure of levels in the design document. For example if there are three nested reflevels in the TOC template, then the design document must have at least three nested levels. In this example we have a single reflevel to correspond to the single level in the design document. It is within the reflevel that the TOC Reference is placed. And it is this TOC Reference 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 design, outside the main template and after it, consists of global templates.

The TOC definitions (TOC Levels and 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 separating chapters from sections and from sub-sections. Each kind of `topic` element (chapter, section, and sub-section) is placed in a different conditional branch so that it can be processed differently than the other types of `topic` element.

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

 © 2012 Altova GmbH

The *Group* property sets the TOC bookmark group. In our example, we have specified the value `toc` 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 (reflevel) has been inserted. This reflevel 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 reflevel, a TOC reference (TOCref) has been inserted. This TOCref has been set to select bookmarks (i) that are in the bookmark group named `toc` (*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 TOCef 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.

Example: Hierarchical and Sequential TOCs

An example SPS file to demonstrate the use of TOCs, called `Chapters.sps`, is in the (My) Documents folder, `C:/Documents and Settings/<username>/My Documents/Altova/StyleVision2012/StyleVisionExamples\Tutorial\TOC`. This SPS is based on a schema that defines the content model of a large chapter-based document. The schema structure is shown in the screenshot below and can be viewed in the Schema Tree window of StyleVision when you open `Chapters.sps`.

The document element is `helpproject`, which contains a child `topics` element. The `topics` element can contain an unlimited number of `topic` elements, each of which can in turn contain descendant `topic` elements. The first level of `topic` elements can be considered to be the chapters of the document, while descendant `topic` elements are sections, sub-sections, and so on.

The SPS contains three TOCs, located at the top of the document, in the following order:

1. *Chapters at a glance*, which lists the names of each chapter (the first-level topics).
2. *Chapters and their sections*, which lists each chapter with its descendants sections (first-level topics, plus each topic's hierarchy of sub-topics down to the lowest-level topic, which in the accompanying XML document, `chapters.xml`, is the third-level topic)
3. *List of images*, which is a flat list of all images in the document (except the first), listed by file name.

 © 2012 Altova GmbH

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 level has been created on the `topic` element, and a bookmark has been created within the `header` child element (but outside the `para` element).

Since the `topic` element is recursive, the levels and the bookmark will also recurse. This means that a new hierarchically subordinate level and and a new bookmark is created for each descendant topic. Since the formatting of the header (the topic title) for each 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 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 *Text from* Value field, 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 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 reflevel, and the TOCref within that reflevel 'toc' has been set to select TOC bookmarks named `toc` 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 reflevels 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).

 © 2012 Altova GmbH

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 reflevel. 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 reflevel, 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. These numbers can also be formatted.

Auto-Numbering

Repeating instances of a node can be numbered automatically using StyleVision's 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. Auto-Numbering can be either flat or hierarchical, and there is a wide variety of formatting available for the numbers.

Flat (sequential) numbering

Flat numbering can be inserted within a TOC Reference or within a TOC Bookmark. Within a TOC Reference, flat numbering will point back to a bookmark in the document; it would be a TOC entry and is described in the Table of Contents (TOC) section. In this section, we describe how to create flat numbering within a TOC Bookmark in the document. Such numbering can be independent of a TOC. To create flat numbering in a document, do the following:

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.
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 flat (sequential) numbering, `( num-seq)` (*highlighted within the TOC Bookmark 'TopicHeader' in the screenshot below*).

3. Right-click the TOC Bookmark and toggle off the command **Construct Entry Text Using XPath**. This is because: (i) the TOC Bookmark is being used solely for flat numbering and not for TOC entries; no text entry for TOC entries is required; and (ii) to ensure that no faulty XPath expression—that can cause a transformation error—is used.
4. There is no need to name the TOC Bookmark (since it will not be referenced from a TOC template), but you could name it if you wish (right-click the TOC Bookmark and select the **Edit Name** command).

In the example shown in the screenshot above, flat 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.

 © 2012 Altova GmbH

Note: If the flat numbering must be continued on another set of nodes, then use a TOC Bookmark that has the same name as that of the TOC Bookmark from which the numbering is to be continued.

Hierarchical numbering

Hierarchical numbering can be inserted within a Reflevel or within a Level in the design. Within a Reflevel, hierarchical numbering will point back to a TOC bookmark in the document; it would be a TOC entry and is described in the Table of Contents (TOC) section. In this section, we describe how to create hierarchical numbering within levels in the document. Such hierarchical numbering can be independent of a TOC.

To create hierarchical numbering in a document, you must first structure the document in levels and create levels as described in the section Structuring the Design in Levels. The following points should be borne in mind:

- Levels must be created either on the node to be numbered or within it.
- Levels must be nested according to the hierarchy of the numbering required (*see screenshot below*).
- The hierarchical numbering placeholder must be inserted within the corresponding level in teh 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 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 | Sequential 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.

Formatting
To format the Auto-Numbering, right-click the Auto-Numbering placeholder (`( num-seq)` or `( num-lvl)`) and select Edit Format. This pops up the respective dialogs (screenshots below), in which you can select the required formatting from a range of options.

- Sequential numbering: Select a numbering style. The selection is is displayed in the Format String box and can be modified there if required.

- Hierarchical numbering: First select the numbering style and then the number of tokens. The resulting format string is displayed in the Format String box. Levels can be omitted by entering the required number of levels to be omitted in the Omit Levels box.

 © 2012 Altova GmbH

Click **OK** when done.

Text References

Anchors can be created on nodes in a document and can be given dynamic names. These anchors can then be referenced by their dynamic names. This means, in effect, that text can be marked for referencing and then referenced from elsewhere in the document

In the GUI, these anchors with dynamic names (the text references) are created by means of TOC Bookmarks, which can use XPath expressions to dynamically locate the text to be referenced. The design can then contain TOC References that identify the required TOC Bookmarks by their names. In this way, the TOC Reference identifies the teext reference and links to it.

TOC References can be located within Levels and within Reference Levels. The latter case is used in TOCs and is described in the section Table of Contents (TOC). In this section, we describe how references are created within levels in the design document, thus enabling them to be used as cross-references.

Step 1: Levels
The document is structured into levels as described in the section Structuring the Design in Levels. The levels will be used during referencing to specify the scope of the referencing. In teh 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 with a name and an XPath expression that identifies the node in the document, the contents of which is the text reference to be located. In the screenshot below, the TOC Bookmark within the `n1:Name` element 'toc3' 'toc3' has a name of `toc3` and an XPath expression that locates the current node. This means that the text reference will be the contents of the `n1:Name` node.

When the XML document is processed, for every `n1:Office/n1:Name` element an anchor is created with a text reference that is the value of the `n1:Office/n1:Name` element.

A TOC Bookmark is inserted in the document by placing the cursor at the required location, right-clicking, and selecting **Insert Table of Contents | TOC Bookmark**.

Step 3: Creating TOC References
A TOC Reference is inserted (context menu, **Insert Table of Contents | TOC Reference**) to create a link to text references generated by a TOC Bookmark. In the screenshot above, the selected location of 'toc3' (text ref) 'toc3' is within the same level as that in which the TOC Bookmark was created (the `Office` level). When defining the TOC Reference, you specify two things. First, the name of the TOC Bookmark to point to; in the case of the screenshot above, the name is `toc3`. Second, the scope of the referencing; in the example shown above, the scope is the current level. This means that TOC Bookmarks within the current level are targeted by this reference.

 © 2012 Altova GmbH

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

The purple text is that generated by the `( text ref)` placeholder of the TOC Reference. The content of the text reference is derived from the XPath expression in the TOC Bookmark referenced by the TOC Reference.

In the above example, 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.

Bookmarks and Hyperlinks

In the SPS document, bookmarks can be inserted anywhere within the design. These bookmarks are transformed into anchors in the output, which can be linked to from hyperlinks. Hyperlinks can not only link to bookmarks, but also to external resources like Web pages. StyleVision offers considerable flexibility in the way target URIs for hyperlinks can be built.

In this section, we describe:

- How bookmarks can be inserted in the SPS.
- How hyperlinks can be inserted in the SPS and how they link to the target pages.

Note: Links to external documents are supported in the FO spec but might not be supported by the FO processor you are using. You should check support for this feature if you are planning to use it.

 © 2012 Altova GmbH

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.

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 content 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 below*). To reference the bookmark, the same ID can be generated as the `href` value of a hyperlink.

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 | Bookmark**, or right-click and select **Insert | Bookmark**.
3. In the Insert Bookmark dialog (*screenshot below*), select a tab according to whether the name of the bookmark should be static (Static tab), dynamically obtained from the XML document or arbitrarily generated from an XPath expression (Dynamic), or composed of both static and dynamic parts (Static and Dynamic). In the screenshot below a dynamic bookmark is created, which has a name that is a unique ID for each `Name` child of the context node.

4. Click **OK**. The bookmark is defined.

After a bookmark has been created, it can be linked to by a hyperlink.

Note: Bookmarks are created at the location specified in the design. If that location is within an element that repeats, a bookmark is created within each instance of that repeating element. If a static name is given, then each bookmark will have the same name. Therefore, it is better in such cases (of repeating elements) to give a dynamic name, which can be, for example, the name of a child element of the context node (the

element within which the bookmark is created). If the node selected for the dynamic name might have the same content across multiple instances, then the uniqueness of the bookmark name can be ensured by using the `generate-id()` function to generate the name (*see screenshot above*). To reference such a bookmark, the same ID can be generated as the `href` value of a hyperlink. In this case make sure you use the fragment-identifier `#` in front of the `generate-id()` function. The XPath expression would be: `concat('#', generate-id(nodeXXX)).`

 © 2012 Altova GmbH

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.

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** or **Insert | Hyperlink**.

The target of a link can be either:

- A bookmark in the same SPS design (in which case the target URI must be a fragment identifier),
- Dynamically generated to match bookmark anchors (these URIs are also fragment identifiers),
- An external resource; the URI can be static (directly entered), dynamic (taken from a node in an XML document), a combination of static and dynamic parts, or the value of an unparsed entity.

How these targets are defined is explained below. After the URI has been defined in the Insert/Edit Hyperlink dialog, click **OK** to finish.

Linking to bookmarks

To link to a bookmark, do the following:

1. In the Static tab of the Insert Hyperlink dialog, click the **Bookmark** button. This pops up

 © 2012 Altova GmbH

the Select Bookmark in Document dialog (*screenshot below*). The screenshot below shows two bookmarks: one static, one dynamic.

2. To select a static bookmark as the target URI, double-click the static bookmark and click **OK**. If you double-click a dynamic bookmark, you will be prompted to enter an XPath expression to match the selected dynamic bookmark (*see screenshot below*).

The dynamic bookmark is actually an XPath expression that generates the name of the bookmark; it is not itself the name of the bookmark. The Create Hyperlink to Dynamic Bookmark dialog, displays the XPath expression of the dynamic bookmark and enables you to construct an XPath expression that will generate a name to match that of the targeted bookmark. Click **OK** when done.

Linking to dynamically generated ID bookmarks
Bookmarks can have dynamically generated ID anchors. If one wishes to link back to such a bookmark, the problem then is this: Since the names of dynamically generated anchors are generated at runtime and therefore unknown at design time, how is one to set the `href` value of a hyperlink that targets such an anchor? The answer is to use the `generate-id()` function once again, this time within the `href` value of the hyperlink. The key to understanding why this works lies in a property of the `generate-id()` function. In a single transformation, each time the `generate-id()` function is evaluated for a specific node, it always generates the same ID. Because of this the IDs generated in the bookmark and the hyperlink will be the same.

Two points should be borne in mind:

- Since the `generate-id()` function must be evaluated as an XPath expression, use the Dynamic tab of the Insert Hyperlink dialog (*see screenshot below*) to set the target of the hyperlink.
- The evaluated value of the `href` attribute must start with `#` (the fragment identifier). Consequently the XPath expression will be: `concat('#', generate-id(nodeXXX))`. Alternatively, in the Static and Dynamic tab, enter `#` in the static part of the address and `generate-id(nodeXXX)` in the dynamic part.

Linking to external resources
URIs that locate external resources can be built in the following ways:

- By entering the URI directly in the Static tab of the Insert Hyperlink dialog. For example, a link to the Altova home page (`http://www.altova.com`) can be entered directly in the Address input field of the Static tab.
- By selecting a node in the XML document source in the Dynamic tab of the Insert Hyperlink dialog. The node in the XML source can provide a text string that is either: (i) the URI to be targeted, or (ii) the name of an unparsed entity which has the required URI as its value. For example, the Altova website address can be contained as a text string in a node.
- By building a URI that has both static and dynamic parts in the Static and Dynamic tab of the Insert Hyperlink dialog. This can be useful for adding static prefixes (e.g. a protocol) or suffixes (e.g. a domain name). For example, email addresses can be created using a static part of `mailto:` and a dynamic part that takes the string content of the `//Contact/@email` node (*screenshot below*).

How to use unparsed entities is described in the section Unparsed Entity URIs.

Editing hyperlink properties

To edit a hyperlink, right-click either the start or end hyperlink (`A`) tag, and select Hyperlink Properties 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.

10.7 Example: Multiple Languages

Very often, documents will need to contain content in multiple languages or will require the user to choose a preferred language. StyleVision offers a range of features that can be used to achieve these goals. Given below are some possibilities, all of which which are demonstrated in the *Multiple Language* examples in the `Examples` project delivered with StyleVision. (The `Examples` project should load automatically by default when you first start StyleVision. It can also be loaded by selecting the menu command **Project | Open**, and then browsing for the `Examples.svp` file in the folder: `C:/Documents and Settings/<username>/My Documents/Altova/StyleVision2012/StyleVisionExamples`.)

Using variables and conditions

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

In the screenshot above, notice that the user's choice is entered as the value of the editable variable. The conditions in the table have two branches for the two language choices and test for the value of he editable variable. The Authentic View output is as in the screenshot below.

The above strategy is well-suited for forms in which the user selects the required language. For

 © 2012 Altova GmbH

details, see the file, `MultiLangByCondition.sps`, which is in the `Examples` project.

Using parameters and Auto-Calculations

Another scenario would be one in which the same data is required to be output in different languages. A possible strategy for this requirement would be to use a parameter, the value of which triggers the required language output. The appropriate language output can be determined, for example, by means of an Auto-Calculation. The Auto-Calculation could output the appropriate content according to the value of the parameter.

In the screenshot above, the Auto-Calcs have XPath expressions of the form:

```
if ( $Language = 'E' ) then 'First' else
if ( $Language = 'G' ) then 'Vorname' else ''
```

The value of the `$Language` global parameter can be modified in the SPS design or can be supplied via the command line at runtime. Multiple transformation runs can be made to output the same data in multiple languages.

For details, see the file, `MultiLangByAutoCalcs.sps`, which is in the `Examples` project.

Example files

For more examples, open the `Examples` project file, `Examples.svp`, which is in the folder: `C:/Documents and Settings/<username>/My Documents/Altova/StyleVision2012/StyleVisionExamples`.

Chapter 11

SPS File: Presentation

11 SPS File: Presentation

In the SPS design, a single set of styling features is defined for components. These styles are converted to the corresponding style markup in the respective outputs (*Authentic View*, *HTML, RTF, PDF and Word 2007+ in the Enterprise Edition; Authentic View, HTML and RTF in the Professional Edition; HTML in the Basic Edition*).

Note: Printed-page based output formats, such as RTF, PDF, and Word 2007+ (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 output) or Enterprise Edition (includes RTF, PDF, and Word 2007+). 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 sub-section of this section.

The values of style properties can be entered directly in the Styles or Properties sidebars, or they can be set via XPath expressions. The benefits of using XPath expressions are: (i) that the property value can taken from an XML file, and (ii) that a property value can be assigned conditionally according to a test contained in the XPath expression.

Additionally, in the SPS design, certain HTML elements are available as markup for SPS components. These predefined formats are passed to the HTML output. The formatting inherent in such markup is therefore also used to provide styling to SPS components. When CSS styles are applied to predefined formats, the CSS styles get priority over the inherent style of the predefined format. Predefined formats are described in the Predefined Formats sub-section of this section.

 © 2012 Altova GmbH

11.1 Predefined Formats

StyleVision provides a number of pre-defined formats, each of which corresponds to an HTML element (*screenshot below*). When you apply a Predefined Format to a component in the Design, that component is marked up as a component having the corresponding HTML semantics. This has two effects:

- Formatting inherent to the selected predefined format is applied.
- The component is contained in the component type, *paragraph*, which makes it available for local styling by component type.

Assigning Predefined Formats

Predefined formats can be assigned by clicking **Insert | Special Paragraph**, and then the required format, or by selecting the required format from the Format drop-down list in the Toolbar (shown below).

Inherent styles

The predefined formats used in StyleVision have either one or both of the following two styling components:

- a text-styling component
- a spacing component.

For example, the predefined `para (p)` format has a spacing component only; it puts vertical space before and after the selected component, and does not apply any text styling. On the other hand, the predefined `Heading 1 (h1)` format has both a text-styling component and a spacing component.

The following styling points about predefined formats should be noted:

- The spacing component of a predefined format applies for any type of SPS component, but the text styling only if it can be applied. For example, if you select an image and apply a predefined format of `Heading 1 (h1)` to it, then the spacing component will take effect, but the text-styling component will not.
- The text-styling component of predefined formats does not apply to data-entry devices.
- Only one predefined format applies to a component at any given time.
- The `Preformatted` predefined format (`pre`) applies formatting equivalent to that applied by the `pre` tab of HTML: linebreaks and spacing in the text are maintained and a monospaced font (such as Courier) is used for the display. In the case of run-on lines with no linebreaks, such as in a paragraph of text, the `Preformatted (pre)` predefined format will display lines of text without wrapping. If you wish to wrap the text, use the predefined format `Preformatted, wrapping (pre-wrap)`.

Defining additional styling for a predefined format

Styles additional to the inherent styling can be defined for a predefined format by selecting it and applying a local style via the Styles sidebar.

© 2012 Altova GmbH

11.2 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).

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	disable-output-escaping	Output text
`&`	no	`&`
`&`	yes	`&`
`&`	no	`&`

&	yes	&
<	no	<
<	yes	<
A	no	A
A	yes	A
<	no	<
<	yes	<
<	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.

 © 2012 Altova GmbH

11.3 Value Formatting (Formatting Numeric Datatypes)

Value Formatting enables the contents of numeric XML Schema datatype nodes (*see list below*) to be displayed in a format other than the lexical representation of that datatype. (For example, the lexical representation of an `xs:date` datatype node is `YYYY-MM-DD`, with an optional timezone component, such as `+02:00`.) The Value Formatting is displayed in 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*) for which Value Formatting is available.

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

- how the Value Formatting mechanism works, and
- the syntax for defining the Value Formatting.

Note: Value Formatting does not change the format in which the data is stored in the XML document. In the valid XML document, the data is always stored in the lexical format appropriate to the datatype of the node. Value Formatting is applied to the display in 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.

The Value Formatting Mechanism

Value Formatting can be applied to:

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

Defining Value Formatting

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

1. Select the `contents` placeholder or input field of the node, or the Auto-Calculation.
2. In the Properties sidebar, select the item, and then the *Content* group (or *AutoCalc* group) of properties. Now click the Edit button ... of the `Value Formatting` property. Alternatively, right-click and select **Edit Value Formatting** from the context menu. The Value Formatting dialog appears (*screenshot below*). It is different according to whether the selected component was a node or an Auto-Calculation. If the selected component was a node, then a dialog like the one below appears. The node represented in the screenshot below is of the `xs:date` datatype.

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.

 © 2012 Altova GmbH

3. You now specify whether the display of the component's value is to be unformatted or formatted. If you wish to leave the output unformatted, select the *Unformatted* radio button. Otherwise select the *Format as XML Schema Value* radio button. (If the value is unformatted, the output has the standard formatting for the datatype of the selected node or the datatype of the Auto-Calculation result. If you specify *Formatting as XML Schema Value* for an Auto-Calculation, you have to additionally select (from a dropdown list) the datatype of the expected Auto-calculation result.
4. Enter the Value Formatting definition. This definition can be entered in three ways: (i) by selecting from a dropdown list of available options for that datatype (*see the 'Format in Output Documents' input field in the screenshots above*); (ii) by entering the definition directly in the input field; and (iii) by using the **Insert Field** and **Field Options** buttons to build the definition correctly. See Value Formatting Syntax for a full description of the various formatting options.

Errors in syntax
If there is an error in syntax, the following happens:

- The definition is displayed in red.
- An error message, also in red, is displayed below the input field.
- The **OK** button in the Value Formatting dialog is disabled.
- The **Go to Error** button in the Value Formatting dialog is enabled. Clicking it causes the cursor to be placed at the point in the format definition where the syntax error is.

Mismatch of data and datatype formats
If the data entered in an XML node does not match the lexical format of that node's datatype, or if the result of an Auto-Calculation does not match the lexical format of the expected datatype, then the formatting will be undefined and will not be displayed correctly in the output.

Applying Value Formatting to the output

The Value Formatting that you define applies to Authentic View, which is supported in the Enterprise and Professional editions.

Some Value Formatting definitions—not all—can also, additionally, be applied to HTML 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).

 © 2012 Altova GmbH

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 e xample, 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
,	Digit 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

 © 2012 Altova GmbH

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)

 © 2012 Altova GmbH

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 (*see note below*)
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.

11.4 Working with CSS Styles

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

- In external CSS stylesheets. External CSS stylesheets can be added via the Design Overview sidebar and via the Style Repository sidebar.
- In global styles for the SPS, which can be considered to be defined within the SPS and at its start. (In the HTML output these global styles are defined within the `style` child element of the `head` element.) Global styles are defined in the Style Repository sidebar.
- Locally, on individual components of the document. In the HTML output, such rules are defined in the `style` attribute of individual HTML elements. Local style rules are defined in the Styles sidebar.

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

Terminology

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

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

or

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

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

Cascading order

The cascading order of CSS applies. This means that precedence of rules are evaluated on the basis of:

1. ***Origin.*** External stylesheets have lower precedence than global styles, and global styles have lower precedence than local styles. External stylesheets are considered to be imported, and the import order is significant, with the latter of two imported stylesheets having precedence.
2. ***Specificity.*** If two rules apply to the same element, the rule with the more specific selector has precedence.
3. ***Order.*** If two rules have the same origin and specificity, the rule that occurs later in the stylesheet has precedence. Imported stylesheets are considered to come before the rule set of the importing stylesheet.

CSS styles in modular SPSs

When an SPS module is added to another SPS, then the CSS styles in the referring SPS have priority over those in the added module. When multiple modules are added, then CSS styles in those modules located relatively lower in the module list have priority. For more information about modular SPSs, see the section, Modular SPSs.

CSS support in Internet Explorer

Versions of Internet Explorer (IE) prior to IE 6.0 interpret certain CSS rules differently than IE 6.0 and later. As a designer, it is important to know for which version of IE you will be designing. IE 6.0 and later offers support for both the older and newer interpretations, thus enabling you to

 © 2012 Altova GmbH

use even the older interpretation in the newer versions (IE 6.0 and later). Which interpretation is used by IE 6.0 and later is determined by a switch in the HTML document code. In an SPS, you can specify whether the HTML output documents should be styled according to Internet Explorer's older or newer interpretation. You should then set CSS styles according to the selected interpretation. For more details, see Properties: CSS Support.

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

External Stylesheets

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

- Adding an external CSS stylesheet to the SPS
- Viewing the contents of an external CSS stylesheet and modifying the media applicability
- Changing the precedence
- Switching between the full CSS stylesheet set and a single CSS stylesheet

External CSS stylesheets can be managed from two sidebars: the Style Repository sidebar and the Design Overview sidebar. If an aspect of the external stylesheets is viewable in both sidebars (for example, the relative precedence of multiple stylesheets), then changes made in one sidebar will automatically be reflected in the other.

Adding an external CSS stylesheet to the SPS

To assign an external CSS stylesheet to the SPS, do the following:

1. In Design View, select the External item in the Style Repository window (*screenshot below*).

2. Click the **Add** button at the top left of the Style Repository toolbar (*see screenshot above*).
3. In the Open dialog that pops up, browse for and select the required CSS file, then click **Open**. The CSS file is added to the External item as part of its tree structure (*see tree listing and screenshot below*).
4. To add an additional external CSS stylesheet, repeat steps 1 to 3. The new CSS stylesheet will be added to the External tree, below all previously added external CSS stylesheets.

Note: You can also add an external CSS stylesheet via the Design Overview sidebar.

Viewing and modifying the tree of external CSS stylesheets

The tree of external CSS stylesheets is structured as follows (*also see screenshot below*):

```
- CSS-1.css  (File location appears on mouseover)
  - Media (can be defined in Style Repository window)
  - Rules (non-editable; must be edited in CSS file)
    - Selector-1
      - Property-1
      - ...
      - Property-N
    - ...
    - Selector-N
+ ...
+ CSS-N.css
```

 © 2012 Altova GmbH

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., however, only class selector rules from the selected CSS file are applied. Other rules are applied from all CSS files. These rules can be applied to the following design components: Auto-Calculations, the (contents) placeholder, paragraph (block) components, and table cells.

 © 2012 Altova GmbH

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, either: (i) right-click and select an option from the Add Selector submenu, or (ii) click the selector and edit it.

4. Now set the CSS property values for the selector. How to do this is explained in the section Setting Style Values.
5. To add another global style, repeat steps 1 to 4. The new global style will be added to the Global tree, below all previously added global styles.

Note:

- Global styles can also be inserted before a selected global style in the Global tree by clicking the **Insert** button in the Style Repository window. The **Add** and **Insert** buttons are also available via the context menu that appears when you right-click a global style or the Global item in the Style Repository window.
- A global style with a selector that is an HTML element can be inserted by right-clicking an item in the Global tree, then selecting **Add Selector | HTML |** ***HTMLElementName***.

Editing and deleting global styles
Both, a style's selector as well as its properties can be edited in the Style Repository window.

- To edit a selector, double-click the selector name, then place the cursor in the text field, and edit.
- For information about defining and editing a style's property values, see Setting Style Values. (The style properties can be displayed in three possible views. These views and how to switch between them are described in Views of Definitions.

To delete a global style, select the style and click the **Reset** button in the Style Repository toolbar.

Changing the precedence of global styles
Global styles that are assigned in the Style Repository window are placed as CSS rules in the `/html/head/style` element. In the HTML file, they would look something like this:

```
<html>
   <head>
      <style>
         <!--
         h1     {  color: blue;
                   font-size: 16pt;
                }
         h2     {  color: blue;
                   font-size: 14pt;
                }
         .main  {  color: green;}
         -->
      </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="main">`, then two global styles match this element: that with the `h1` selector and that with the `.main` selector. The color property of `.main` selector will apply because it occurs after the `h1` selector in the style listing. 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 `.main` global style were moved to a position before

 © 2012 Altova GmbH

the `h1` style, then the color property of the `h1` style would have precedence over that of the `.main` style.

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 in the Styles sidebar. (This is as opposed to global styles, which are defined in the Style Repository sidebar.)

Defining a style locally consists of two 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.

 © 2012 Altova GmbH

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* column. 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* column, 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 column. How to do this is described in the section Setting Style Values.

Setting Style Values

For the component-type selected in the *Styles-For* column, style properties are defined in the Style Definitions column of the Styles sidebar (*screenshot below*) . You can select more than one component-type in the *Styles-For* column if you like—by selecting additional component-types with the **Ctrl**-key pressed, or by selecting a range of component-types in the *Styles-For* column with the **Shift**-key pressed. When multiple component-types are selected, any style value you define in the *Style-Definitions* column is applied to all instances of all the selected component-types.

Style property groups
The available style properties in the *Style-Definitions* column are organized into groups as shown in the screenshot below.

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

 © 2012 Altova GmbH

Entering style values
Style property values (style values for short) can be entered in the following ways, all of which are show in the screenshot below:

- Entered directly in the Value column. To do this, select a property, double-click in its Value column, enter the value using the keyboard, and press **Enter** or click anywhere in the GUI.
- By selecting a value from the dropdown list of the combo box for that property. Click the down arrow of the combo box to drop down the list of style-value options. In the screenshot below, the options for the `(background-)repeat` property are displayed. Select the required value from the dropdown list.
- By using the icon on the right-hand side of the Value column for that style property. Two types of icon are available, and these are available only for properties to which they are relevant: (i) a color palette for selecting colors (in the screenshot below, see the `(background-)color` property), and (ii) a dialog for browsing for files (in the screenshot below, see the `(background-)image` property).

- Values for styles can also be assigned via an XPath expression.

Modifying or deleting a style value
If a style value is entered incorrectly or is invalid, it is displayed in red. To modify the value, use any of the applicable methods described in the previous section, Entering Property Values.

To delete a style value (or, in other words, to reset a style value, click the Reset button in the toolbar of the Styles sidebar. Alternatively, you can double-click in the Value column of the property, and delete the value using the **Delete** and/or **Backspace** key, and then pressing **Enter**.

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, 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, two different background colors can be selected depending on the position of an element in a sequence. Thus, when these elements are presented as rows in a table, the odd-numbered rows can be presented with one background color while the even-numbered rows are presented with another (*see below for example*). Also, depending on the content of a node, the presentation can be varied.

Style properties for which XPath expressions are enabled
XPath expressions can be entered for the following style properties:

- All properties available in the Styles sidebar
- The *Common*, *Event*, and *HTML* groups of properties in the Properties sidebar

Static mode and dynamic (XPath) mode for property values
For those properties where XPath expressions are enabled, two mode are available:

- Static mode, where the value of the property is entered directly in the Value column of the sidebar. For example, for the background-color of a design component, the value `red` can be entered directly in the sidebar.
- Dynamic, or XPath mode, where an XPath expression is entered. The expression is evaluated at runtime, and the result is entered as the value of the property. For example, for the background color of a design component, the following XPath expression can be entered: `/root/colors/color1`. At runtime, the content of the node `/root/colors/color1` will be retrieved and entered as the value of the background-color property.

Switching between static and dynamic (XPath) modes
For each property for which XPath expressions are enabled, static mode is selected by default. To switch a property to dynamic (XPath) mode, select that property and click the XPath icon in the toolbar of the sidebar (*screenshot below*).

 © 2012 Altova GmbH

If a static value was present for that property, it is now cleared and the mode is switched to dynamic. The Edit XPath Expression dialog appears. It is in this dialog that you enter the XPath expression for the property. Click **OK** when finished.

After you enter an XPath expression for the property, an Edit XPath expression button appears in the Value column for that property (*screenshot above*). Click this button to subsequently edit the XPath expression. If you wish to switch back to static mode, click the XPath icon in the toolbar. This will clear the XPath expression and switch the property to static mode.

Note: There are two important points to note. First, only one mode (static or dynamic), and the value/expression for that mode, is active at any time. Any value/expression that previously existed for the other mode is cleared; so switching to the other mode will present that mode with an empty entry field. (In order to go back to a previous value/expression, use the Undo command.) Second, if you reselect a property after further editing the SPS, then that property will be opened in the mode it was in previously.

Creating and editing the XPath definition

The XPath definition is created and edited in the Edit XPath Expression dialog. This dialog is accessed in two ways:

- Each time you switch to the dynamic mode of a property from static mode (by clicking the XPath icon in the toolbar of the sidebar), the Edit XPath Expression dialog appears. You can now create the XPath expression. (Note that clicking the toolbar icon when already in dynamic mode switches the mode to static mode; it does not pop up the Edit XPath Expression dialog.)
- The Edit XPath Expression dialog also pops up when you click the Edit XPath Expression button in the Value field of a property that already has an XPath expression defined for it. The dialog will contain the already defined XPath expression for that property, which you can now edit.

After you enter or edit the XPath expression in the entry field, click **OK** to finish.

Values returned by XPath expressions
The most important benefits of using XPath expressions to set a property value are that: (i) the property value can be taken from an XML file (instead of being directly entered); and/or (ii) an XPath expression can test some condition relating to the content or structure of the XML document being processed, and accordingly select a value. XPath expressions return values in the following two categories:

- *XML node content*
  The XPath expression can address nodes in: (i) the XML document being processed by the SPS, or (ii) any accessible XML document. For example the expression `Format/@color` would access the `color` attribute of the `Format` child of the context node. The value of the `color` attribute will be set as the value of the property for which the XPath expression was defined. A node in some other XML document can be accessed using the `doc()` function of XPath 2.0. For example, the expression `doc('Styles.xml')//colors/color-3` would retrieve the value of the element `color-3` in the XML document `Styles.xml` and set this value as the value of the property for which the XPath expression was defined.

- *XPath expression*
  The value of the property can come from the XPath expression itself, not from the XML document. For example, the background color of an element that is being output as a row can be made to alternate depending on whether the position of the row is odd-numbered or even-numbered. This could be achieved using the XPath 2.0 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.

 © 2012 Altova GmbH

Composite Styles

A Composite Style is a group of CSS text-styling properties that have been associated with an attribute of an XML instance document node. Additionally, any group of CSS text-styling properties stored in the stylesheet is also considered to be a Composite Style. Composite Styles can then be specified on the following design components:

- Auto-Calculations
- The (contents) placeholder
- Paragraph (block) design elements
- Table cells

Advantages of Composite Styles

Composite Styles offer the following advantages:

- Styling properties are in the XML data and can therefore be edited by the user.
- 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 dialog, 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:

 © 2012 Altova GmbH

```
<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.

Chapter 12

SPS File: Additional Functionality

12 SPS File: Additional Functionality

Additional to the content editing, structure, advanced, and presentation procedures described in this documentation, StyleVision provides a range of miscellaneous additional features. These are listed below and described in detail in the sub-sections of this section.

- Global Resources. Global resources provide flexibility in selecting resources. For example, multiple resources (such as files), can be assigned to an alias. When an alias is used as a source (XML, XSD, etc) of an SPS, the resource can be switched among the multiple resources assigned to the alias.
- Working with Dates.Dates can be manipulated and formatted as required.
- Unparsed Entity URIs. URIs can be stored in unparsed entities in the DTD on which an XML document is based. The Unparsed Entity URI feature enables images and hyperlinks to use these URIs as target URIs.
- Using Scripts. StyleVision contains a JavaScript Editor in which JavaScript functions can be defined. These functions are then available for use as event handlers anywhere within the SPS, and will take effect in the output HTML document.
- HTML Import. An HTML file can be imported into StyleVision and an XML, XSD, and SPS files can be created from it.
- New from XSLT. An SPS can be created from an XSLT-for-HTML or an XSLT-for-FO. Template structure and styling in the XSLT will be created in the SPS. You can then modify the SPS components and add content and formatting to the SPS.

 © 2012 Altova GmbH

12.1 Altova Global Resources

Altova Global Resources is a collection of aliases for file, folder, and database resources. Each alias can have multiple configurations, and each configuration maps to a single resource

Therefore, when a global resource is used as an input, the global resource can be switched among its configurations. This is done easily via controls in the GUI. For example, if an XSLT stylesheet for transforming an XML document is assigned via a global resource, then we can set up multiple configurations for the global resource, each of which points to a different XSLT file. After setting up the global resource in this way, switching the configuration would switch the XSLT file used for the transformation.

A global resource can not only be used to switch resources within an Altova application, but also to generate and use resources from other Altova applications. So, files can be generated on-the-fly in one Altova application for use in another Altova application. All of this tremendously eases and speeds up development and testing.

Using Altova Global Resources involves two processes:

- Defining Global Resources: Resources are defined and the definitions are stored in an XML file. These resources can be shared across multiple Altova applications.
- Using Global Resources: Within an Altova application, files can be located via a global resource instead of via a file path. The advantage is that the resource being used can be instantly changed by changing the active configuration in StyleVision.

Global resources in other Altova products
Currently, global resources can be defined and used in the following individual Altova products: XMLSpy, StyleVision, MapForce, and DatabaseSpy.

Defining Global Resources

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

- Click Tools in the menu bar to pop up the **Tools** menu (*screenshot below*), and select the command **Global Resources**. This pops up the Global Resources dialog.

- Click the menu command **View | Toolbars | Global Resources** to display the Global Resources Toolbar (*screenshot below*).

  Once the toolbar is displayed, click the Manage Global Resources icon. This pops up the Global Resources dialog.

The Global Resources XML File

Information about global resources that you define is stored in an XML file. By default, this XML file is called `GlobalResources.xml`, and it is stored in the `\Altova\` sub-folder of the (My) documents folder. This file is set as the default Global Resources XML File for all Altova applications. As a result, a global resource defined in any application will be available to all Altova applications—assuming that all applications use this file.

You can also re-name the file and save it to any location, if you wish. Consequently, you may have multiple Global Resources XML files. However, only one of these Global Resources XML File can be active at any time, and only the definitions contained in this file will be available to the application.

To select a Global Resources XML file to be the active file, in the Manage Global Resources dialog (*screenshot below*), browse for it in the Definitions File entry and select it.

 © 2012 Altova GmbH

Managing global resources: adding, editing, deleting
In the Manage Global Resources dialog (*screenshot above*), you can add a global resource to the selected Global Resources XML File, or edit or delete a selected global resource. The Global Resources XML File organizes the aliases you add into a list of several sections: files, folders, and databases (*see screenshot above*).

To add a global resource, click the **Add** button and define the global resource in the **Global Resource** dialog that pops up (*see description below*). After you define a global resource and save it, the global resource (or alias) is added to the library of global definitions in the selected Global Resources XML File. To edit a global resource, select it and click **Edit**. This pops up the **Global Resource** dialog, in which you can make the necessary changes (see the descriptions of files, folders, and databases in the sub-sections of this section). To delete a global resource, select it and click **Delete**.

After you finish adding, editing, or deleting, make sure to click **OK** in the **Manage Global Resources** dialog to save your modifications to the Global Resources XML File.

Adding a global resource
Creating a global resource involves mapping one alias name to one or more resources (file, folder, or database). Each mapping is called a configuration. A single alias name can therefore be associated with several resources via different configurations (*screenshot below*).

In the **Manage Global Resources** dialog (*screenshot above*), when you click the **Add** button, you can select whether you wish to add a file-type, folder-type, or database-type resource. How to add and edit each type of resource is described in the sub-sections of this section.

Files

In the Global Resource dialog for Files (*screenshot below*), you can add a file resource as follows:

1. Enter an alias name.
2. The Configurations pane will have a configuration named Default (*screenshot above*). This Default configuration cannot be deleted nor have its name changed. You can enter as many additional configurations for the selected alias as you like. Add a configuration by clicking the **Add Configuration** icon and, in the **Add Configuration** dialog which pops up, enter the configuration name. Click **OK**. The new configuration will be listed in the Configurations pane. Repeat for as many configurations as required for this particular alias (global resource). You can also copy a configuration (using the Add Configuration as Copy icon) and then modify it.
3. Select one of the configurations in the Configurations pane and then define the resource to which this configuration will map. In the Settings for Configuration X pane, you can select whether the resource is a file, or the result of either an Altova MapForce or Altova StyleVision transformation. After selecting the resource type by clicking its radio button, browse for the file, MapForce file, or StyleVision file. Where multiple inputs or outputs for the transformation are possible, a selection of the options will be presented. For example, if the Result of StyleVision Transformation was selected as the resource type, the output options are displayed according to the what edition of

 © 2012 Altova GmbH

StyleVision is installed (*the screenshot below shows the outputs for Enterprise Edition*).

Select the radio button of the desired option (in the screenshot above, 'HTML output' is selected). The result of a transformation can itself be saved as a global resource or as a file path (click the icon and select, respectively, Global Resource or Browse). If neither of these two saving options is selected, the transformation result will be loaded as a temporary file when the global resource is invoked.

4. Specify a resource for each configuration (that is, repeat Step 3 above for the various configurations you have created).
5. Click **OK** in the Global Resource dialog to save the alias and all its configurations as a global resource. The global resource will be listed under Files in the Manage Global Resources dialog.

Selecting Result of MapForce transformations as a global resource

Altova MapForce maps one or more (already existing) schemas to one or more (new) schemas designed by the MapForce user. XML files corresponding to the input schemas are used as data sources, and an output XML file based on the user-designed schema can be generated by MapForce. This generated output file (Result of MapForce Transformation) is the file that will be used as a global resource.

In a MapForce transformation that has multiple output schemas, you can select which one of the output schemas should be used for the global resource by clicking its radio button (*screenshot below*). The XML file that is generated for this schema can be saved as a global resource or as a file path (click the icon and select, respectively, Global Resource or Browse). If neither of these options is selected, a temporary XML file is created when the global resource is used.

Note that each Input can also be saved as a global resource or as a file path (click the icon and select, respectively, Global Resource or Browse).

 © 2012 Altova GmbH

Folders

In the Global Resource dialog for Folders (*screenshot below*), you can add a folder resource as follows:

Enter an alias name.

1. The Configurations pane will have a configuration named Default (*screenshot above*). This Default configuration cannot be deleted nor have its name changed. You can enter as many additional configurations for the selected alias as you like. Add a configuration by clicking the **Add Configuration** icon and, in the **Add Configuration** dialog which pops up, enter the configuration name. Click **OK**. The new configuration will be listed in the Configurations pane. Repeat for as many configurations as required for this particular alias (global resource).
2. Select one of the configurations in the Configurations pane and browse for the folder you wish to create as a global resource.
3. Specify a folder resource for each configuration (that is, repeat Step 3 above for the various configurations you have created).
4. Click **OK** in the Global Resource dialog to save the alias and all its configurations as a global resource. The global resource will be listed under Folders in the Manage Global Resources dialog.

Copying Configurations

The Manage Global resources dialog allows you to duplicate existing configurations for all types of resources. To do so, select a configuration and click the **Copy Configuration** icon . Then select or enter a configuration name and click **OK**. This creates a copy of the selected configuration which you can now change as required.

© 2012 Altova GmbH

Using Global Resources

There are several types of global resources (file-type, folder-type , and database-type). Particular scenarios in StyleVision allow the use of particular types of global resources. For example, you can use file-type or folder-type global resources for a Working XML File or a CSS file. Or you can use a database-type resource to create a new DB-based SPS. The various scenarios in which you can use global resources in StyleVision are listed in this section: Files and Folders and Databases.

Selections that determine which resource is used
There are two application-wide selections that determine what global resources can be used and which global resources are actually used at any given time:

- *The active Global Resources XML File* is selected in the Global Resource dialog. The global-resource definitions that are present in the active Global Resources XML File are available to all files that are open in the application. Only the definitions in the active Global Resources XML File are available. The active Global Resources XML File can be changed at any time, and the global-resource definitions in the new active file will immediately replace those of the previously active file. The active Global Resources XML File therefore determines: (i) what global resources can be assigned, and (ii) what global resources are available for look-up (for example, if a global resource in one Global Resource XML File is assigned but there is no global resource of that name in the currently active Global Resources XML File, then the assigned global resource (alias) cannot be looked up).
- *The active configuration* is selected via the menu item **Tools | Active Configuration** or via the Global Resources toolbar. Clicking this command (or drop-down list in the toolbar) pops up a list of configurations across all aliases. Selecting a configuration makes that configuration active application-wide. This means that wherever a global resource (or alias) is used, the resource corresponding to the active configuration of each used alias will be loaded. The active configuration is applied to all used aliases. If an alias does not have a configuration with the name of the active configuration, then the default configuration of that alias will be used. The active configuration is not relevant when assigning resources; it is significant only when the resources are actually used.

Assigning Files and Folders

In this section, we describe how file-type and folder-type global resources are assigned. File-type and folder-type global resources are assigned differently. In any one of the usage scenarios below, clicking the **Switch to Global Resources** button pops up the Open Global Resource dialog (*screenshot below*).

Selecting a *file-type global resource* assigns the file. Selecting a *folder-type global resource* causes an Open dialog to open, in which you can brows for the required file. The path to the selected file is entered relative to the folder resource. So if a folder-type global resource were to have two configurations, each pointing to different folders, files having the same name but in different folders could be targeted via the two configurations. This could be useful for testing purposes.

In the Open Global Resource dialog, you can switch to the file dialog or the URL dialog by clicking the respective button at the bottom of the dialog. The **Manage Global Resources** icon in the top right-hand corner pops up the **Manage Global Resources** dialog.

Usage scenarios

File-type and folder-type global resources can be used in the following scenarios:

- Adding and modifying schema sources and Working XML Files
- Saving as Global Resource
- Adding modules and CSS files
- Adding global resources to a project

Schema, Working XML File

In the Design Overview sidebar (*screenshot below*), the context menus for the Schema and Working XML File contains an entry that pops up the Open dialog in which you can assign the schema or Working XML File via a global resource. Clicking the **Switch to Global Resources**

 © 2012 Altova GmbH

button pops up a dialog with a list of all file-type global resources that are defined in the Global Resources XML File currently active in StyleVision. (How to set the currently active Global Resources XML File is described in the section Defining Global Resources.)

If a global resource has been selected as the file source, it is displayed in the relevant entry in the Design Overview sidebar (*screenshot below*).

Adding modules and CSS files from a global resource
In the Design Overview sidebar, the **Add New Module** and **Add New CSS File** commands pop up the Open dialog, in which you can click Switch to Global Resources to select a Global Resource to be used. Modules and CSS files can then be changed by changing the configuration.

Saving as global resource
A newly created file can be saved as a global resource. Also, an already existing file can be opened and then saved as a global resource. When you click the **File | Save** or **File | Save As** commands, the Save dialog appears. Click the **Switch to Global Resource** button to access the available global resources (*screenshot below*), which are the aliases defined in the current Global Resources XML File.

Select an alias and the click Save. If the alias is a file alias, the file will be saved directly. If the alias is a folder alias, a dialog will appear that prompts for the name of the file under which the file is to be saved. In either case the file will be saved to the location that was defined for the currently active configuration.

Note: Each configuration points to a specific file location, which is specified in the definition of that configuration. If the file you are saving as a global resource does not have the same filetype extension as the file at the current file location of the configuration, then there might be editing and validation errors when this global resource is opened in StyleVision. This is because StyleVision will open the file assuming the filetype specified in the definition of the configuration.

Global Resources in projects

Global resources can also be added to the currently active project via the **Project | Add Global Resource to Project** command. This pops up a dialog listing the file-type global resources in the currently active Global Resources XML File. Select a global resource and click **OK** to add it to the project. The global resource appears in the Project sidebar and can be used like any other file.

 © 2012 Altova GmbH

Assigning Databases

When an SPS is created from a database (DB) with the **File | New from DB** command, you can select the option to use a global resource (*screenshot below*).

When you click the Global Resources icon in the Open Database dialog, all the database-type global resources that have been defined in the currently active Global Resources XML File are displayed. Select the required global resource and click **Connect**. If the selected global resource has more than one configuration, then the database resource for the currently active configuration (check **Tools | Active Configuration** or the Global Resources toolbar) is used, and the connection is made. You must now select the data structures and data to be used as described in DB Data Selection.

See also:

Defining Global Resources, for information about defining Global Resources.
Tools | Global Resources, for the menu command to access the Altova Manage Global Resources dialog.
Tools | Active Configuration, for the menu command to change the active configuration of the application.

Changing Configurations

One global resource configuration can be active at any time, and it is active application-wide. This means that the active configuration is active for all aliases in all currently open files. If an alias does not have a configuration with the name of the active configuration, then the default configuration of that alias will be used.

As an example of how to change configurations, consider the case in which a Working XML File has been assigned to an SPS via a global resource with multiple configurations. The Working XML File can be switched merely by changing the configuration of the global resource. This can be done in two ways:

- When you hover over the menu command **Tools | Active Configuration**, a submenu with a list of all configurations in the Global Resources XML File pops out. Select the required configuration.
- In the combo box of the Global Resources toolbar (*screenshot below*), select the required configuration. (The Global Resources toolbar can be toggled on and off with the menu command **View | Toolbars | Global Resources**.)

The Working XML File will be changed immediately.

In this way, by changing the active configuration, you can change source files that are assigned via a global resource. Note, however, that the newly selected files must agree schematically with the SPS in order for the SPS to function correctly.

 © 2012 Altova GmbH

12.2 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, with (i) the URI, and (ii) the notation (which indicates to StyleVision the resource type of the entity).
2. In the XML document, the unparsed entity must be referenced. This is done by giving the names of the required unparsed entities.
3. In the SPS, unparsed entities can be used to target images and hyperlinks by correctly accessing the relevant dynamic node values as unparsed entities.

Declaring and referencing unparsed entities

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

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE document SYSTEM "UEURIDoc.dtd" [
<!ENTITY Picture SYSTEM "nanonull.gif" NDATA GIF>
<!ENTITY AltovaURI SYSTEM "http://www.altova.com" NDATA LNK>
]>
<document>
   <header>Example of How to Use Unparsed Entity URIs</header>
   <para>...</para>
   <img src="Picture"/>
   <link href="AltovaURI">Link to the Altova Website.</link>
</document>
```

SPS images and hyperlinks that use unparsed entities

Images and hyperlinks in the SPS that reference unparsed entity URIs are used as follows:

1. Insert the image or hyperlink via the **Insert** menu.
2. In the Edit dialog of each, select the Dynamic tab properties (*screenshot below*), and enter an XPath expression that selects the node containing the name of the unparsed entity. In the XML document example given above, these nodes would be, respectively, the `//img/@src` and `//link/@href` nodes.

3. Then check the Treat as Unparsed Entity check box at the bottom of the dialog. This causes the content of the selected node to be read as an unparsed entity. If an unparsed entity of that name is declared, the URI associated with that unparsed entity is used to locate the resource (image or hyperlink).

When the stylesheet is processed, the URI associated with the entity name is substituted for the entity name.

Note: Note that if the URI is a relative URI, the XSLT processor expands it to an absolute URI applying the base URI of the DTD. So if the unparsed entity is associated with the relative URI "`nanonull.gif`", then this URI will be expanded to `file:///c:/someFolder/nanonull.gif`, where the DTD is in the folder `someFolder`.

 © 2012 Altova GmbH

12.3 New from XSLT, XSL-FO or FO File

An SPS design can be based on existing XSLT files that were designed for HTML output or XSLT files with XSL-FO commands for output in PDF or FO files. This means that SPS files do not have to be designed from scratch, but can take an already existing XSLT file as a starting point.

Steps for creating an SPS from XSLT
The steps for creating an SPS file from an XSLT, XSLT-for-FO, or FO file are as follows.

1. Select the command **File | New | New from XSLT, XSL-FO or FO File**.
2. In the Open dialog that appears, browse for the file you want.
3. In the next dialog you will be prompted to select a schema on which the SPS is to be based. Select the schema you want.
4. An SPS based on the structure and formatting in the XSLT or FO file will be created and displayed in Design View.
5. You can now modify the SPS in the usual way. For example, you could drag in nodes from the Schema Tree, modify the styling and presentation or add additional styling, and use StyleVision functionality such as Auto-Calculations and Conditional Templates.
6. You can save the SPS and use a Working XML File to preview various output formats. Subsequently you can generate stylesheets and output files using the **Save Generated Files** command.

Example
The example discussed below is located in the (My) Documents folder, `C:/Documents and Settings/<username>/My Documents/Altova/StyleVision2012/StyleVisionExamples/Tutorial/NewFromXSLT`. This folder contains the files: `SimpleExample.xslt`, `SimpleExample.xsd`, and `SimpleEample.xml`.

The XML file is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<Data xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="YearlySales.xsd">
        <ChartType>Pie Chart 2D</ChartType>
        <Region id="Americas">
                <Year id="2005">30000</Year>
                <Year id="2006">90000</Year>
                <Year id="2007">120000</Year>
                <Year id="2008">180000</Year>
                <Year id="2009">140000</Year>
                <Year id="2010">100000</Year>
        </Region>
        <Region id="Europe">
                <Year id="2005">50000</Year>
                <Year id="2006">60000</Year>
                <Year id="2007">80000</Year>
                <Year id="2008">100000</Year>
                <Year id="2009">95000</Year>
                <Year id="2010">80000</Year>
        </Region>
        <Region id="Asia">
                <Year id="2005">10000</Year>
                <Year id="2006">25000</Year>
                <Year id="2007">70000</Year>
                <Year id="2008">110000</Year>
                <Year id="2009">125000</Year>
                <Year id="2010">150000</Year>
        </Region>
</Data>
```

The XSLT file is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:fn="
http://www.w3.org/2005/xpath-functions">
   <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
   <xsl:template match="/">
      <html>
         <head>
            <title>Simple Example for New From XSLT</title>
         </head>
         <body>
            <xsl:apply-templates/>
         </body>
      </html>
   </xsl:template>
   <xsl:template match="Data">
      <xsl:for-each select="Region">
         <h1 style="color:red">
            <xsl:apply-templates select="@id"/>
         </h1>
      </xsl:for-each>
   </xsl:template>
</xsl:stylesheet>
```

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

Notice that the two templates in the XSLT have been created in the SPS. Now switch to the HTML Preview (*screenshot below*), and notice that the `h1` element's styling (`color: red`) has been also passed to the SPS.

 © 2012 Altova GmbH

In Design View select the `h1` element and change its color to black (in the Styles sidebar, in the *Color* group of properties). Then, from the Schema Tree, drag the `Year` element and create it as a table at the location shown in the screenshot below. Reverse the contents of the two columns so that the Year ID is in the first column.

You can make additional changes in the content, structure, and presentation properties of the document, then preview the output and save files using the **Save Generated Files** command.

12.4 User-Defined XPath Functions

The SPS designer can define customized **XPath 2.0** 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.

 © 2012 Altova GmbH

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 expression `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.

 © 2012 Altova GmbH

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. The return datatype is specified to be of the `xs:decimal` type, which is the datatype returned by the `avg()` function when evaluating input values of datatype `xs:integer`. If the return type is specified, then the datatype of the return value is checked to see if it conforms with the specified type. If it doesn't, an error is returned.

Defining the function

To define a function, click the icon of the XPath Functions item in the Schema Tree or Design Tree. This pops up the XPath Functions dialog (*screenshot above*). If you wish to edit a function that has already been created, double-click its entry in the list of XPath functions. Then enter a name for the function and a definition in the Function Body pane. Parameter definitions can be entered if required (see the next two sections, Parameters and Sequences and Parameters and Nodes, for details). A return type for the function can also be specified (*see above*).

The most important point to bear in mind when writing the XPath expression that defines XPath function is that there is **no context node** for the XPath expression. If the XPath expression must locate a node then the context node for the expression can be provided in one of the following ways:

1. The XPath expression starts with the document root. The document root is specified in the first location step of the XPath expression as `$XML`. For example, the XPath expression `$XML/Trades/Stock[1]` locates the first `Stock` child element of the `/Trades` element. The variable `$XML` (which locates the document root of the main schema) is defined globally by StyleVision in all SPS designs.
2. The context node can be passed as a parameter. See the section Parameters and Nodes below for an explanation.

In the following cases, errors are returned:

- If a parameter is defined but is not used in the body of the definition.

 © 2012 Altova GmbH

- If the datatype of the value returned by the function does not match the return type defined for the function.
- If any function in the SPS contains an error, an XSLT error is generated for the whole design, even if the function containing the error is not called. Note, however, that a function can be disabled by unchecking its check box in the list of user-defined XPath functions. When disabled in the design, the function is not included in the XSLT document generated from the design. In this way, an XPath expression containing an error can be excluded from the XSLT and no XSLT error will be generated.

Reusing Functions to Locate Nodes

In the previous section we saw how an XPath function can be built to locate a node. The `sps:Stock` function which is defined as shown in the screenshot below returns the `/Trades/Stock` element that has a `name` attribute with a value that matches the content of the `/Trades/Selection/Stock` element.

We could modularize the location steps of the XPath expression `$XML/Trades/Stock[@name=$XML/Trades/Selection/Stock]` into separate XPath functions. For example as follows:

- The function `sps:Stocks()`, with the definition: `$XML/Trades/Stock`
- The function `sps:SelectedStock()`, with the definition: `$XML/Trades/Selection/Stock`

The whole XPath expression can then be written in another XPath expression as:

```
sps:Stocks()[@name=sps:SelectedStock()]
```

When XPath functions are created in this way to locate a node or nodeset, these functions can be re-used in other XPath expressions across the SPS design, thus considerably simplifying the writing of complex XPath expressions.

 © 2012 Altova GmbH

Parameters in XPath Functions

A function can be assigned any number of parameters. This is done in the Parameters pane of the XPath Functions dialog (*see screenshot below*). Parameters that have been defined can then be used in the definition of the function (in the Function Body pane).

Parameter values are passed to the parameters with the call to function. You should note that parameter values are evaluated before being passed to the function. Errors are returned if: (i) a parameter is defined but not used in the function definition, or (ii) the number of input parameter-values does not match the number of parameters, or (iii) the type of any one input parameter-value doe snot match the type specified for it in the parameter definition.

Order of parameters
The order of the parameters is important because, when the function is called, the parameter-values submitted will be assigned to the parameters according to the order in which they are defined in the Parameters pane (*shown above*). So if the `sps:Stock` XPath function is defined as in the screenshot above and if it is called with the following XPath expression: `sps:Stock($XML, Node1, Node2)`, then these three values—`$XML, Node1, Node2`— will be assigned, respectively, to the parameters `$ContextStock`, `$Selection`, and `$StockInfo`. Note that each input parameter-value is separated from the next by a comma.

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 expected datatype of a parameter can be specified. If a datatype is specified, then the datatype of the input parameter-value will be checked against the specified datatype, and an error will be returned if the types do not match. This feature enables the input data to be checked.

Occurrence
Each parameter can be considered to correspond to a sequence. The *Occurrence* property of each parameter specifies how many items may occur in the sequence submitted for that parameter. Since commas are used both to separate the items in a sequence as well as to separate one parameter (or parameter-value) from another, parentheses are used to delimit parameters (or parameter-values) when required and thus disambiguate an XPath expression. In this context, the following examples and points should be noted:

- Several XPath 2.0 functions take a single sequence as an argument, and this single sequence must be enclosed in parentheses. For example, in the function `avg( (count ($a), $b, $c) )`, the XPath 2.0 `avg()` function takes the sequence `count($a), $b, $c` as its argument, and this sequence must be enclosed in parentheses. The outer pair of parentheses are the delimiters of the function.
- When functions are called, parameter-values are supplied with the call. Parentheses must be correctly used in input parameter-values. For example, if the user-defined XPath function named `MyAverage()` is defined with the XPath 2.0 expression: `avg ( (count($a), $b, $c) )`, then the following input parameter-value would be valid: `MyAverage( (1,2,3), 4, 5 )`. The values corresponding to the three parameters `$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 by parentheses.
- If a datatype is specified for a parameter that can have more-than-one occurrences (that is, more than one item in its sequence), then each item in the input sequence must be of the specified datatype.

 © 2012 Altova GmbH

Parameters and Sequences

It is important to note the relationship between parameters and sequences. These must be considered in two related parts:

- The relationship between parameters and sequences in the definition of the function.
- The relationship between input parameter-values and sequences. The input parameter-value is submitted with the call to the function.

If the function definition is an XPath expression that takes a sequence as an input (*see screenshot below for an example*), you must be careful to use the correct XPath syntax. Parentheses play an important role, indicating that the contents of parentheses are to be evaluated first. In sequences, where commas are used to separate items, parentheses serve to delimit a sequence. A number of XPath 2.0 functions take a sequence as its single argument and operate on the items of the sequence. Here are some examples:

- `avg( (10, 20, 30) )` The `avg` function of XPath 2.0 takes one sequence of items as its input. This sequence, indicated in bold in the example, is delimited by parentheses, and its items are separated by commas. (The outer pair of parentheses are the parentheses of the function.)
- `avg( //Person/@salary )` The location step selects a set of `salary` attribute nodes and returns their attribute-values as the sequence to be evaluated (that is, averaged). No parentheses are required because the sequence is not directly entered. Instead, the required values are obtained from the XML document and submitted to the function as a sequence.
- `count( (10 to 34) )` The parentheses indicate that the range submits a single sequence of items.
- `count( (10 to 34, 37) )` The parentheses indicate that the sequence consists of the items submitted by the range plus the item `37`.
- `count( //Person )` There are no parentheses around the location step that returns the sequence of the evaluated nodeset values.

When **parameters** are used in a function definition that requires a sequence as its argument (*see definition in screenshot below*), then these parameters must be placed within parentheses to indicate the sequence delimitation. Each item in the sequence is separated from the following item by a comma.

Each parameter can considered to be a single sequence. The number of items allowed within the sequence is specified with the Occurrence property. In the definition above, for example, each parameter is defined (in its *Occurrence* property) as a singleton-sequence (that is, a sequence of exactly one item). Each parameter-value must therefore be seen as being a sequence of one item. These singleton-sequences are in turn the items that will constitute the sequence that is the single argument of the function. If any of the input parameter-values is not a singleton-sequence, an error is returned.

Given below are examples of parameter usage in **calls** to the XPath function `ThreeAverage()` shown in the screenshot above.

- `sps:ThreeAverage(10,20,30)` returns `20`. There are three valid input parameter-values, corresponding to the three parameters.
- `sps:ThreeAverage( (10),(20),(30) )` returns `20`. There are three valid input parameter-values, corresponding to the three parameters. Each input parameter-value has been enclosed with parentheses (which are redundant, since each sequence is a singleton-sequence, but not an error).
- `sps:ThreeAverage( (10),20,30 )` returns `20`. There are three valid input parameter-values, corresponding to the three parameters. The first input parameter-value has been enclosed with parentheses (redundant but not an error).
- `sps:ThreeAverage( (10,20),(30),(40) )` returns an error because the first input parameter-value 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 parameter-value is submitted, inside the parentheses. Additionally, the input parameter-

 © 2012 Altova GmbH

value is invalid because the sequence is not a singleton-sequence.

If the *Occurrence* property of a parameter is set to `At-least-one` (*as in the definition shown in the screenshot below*), then that parameter is defined as a sequence of one-or-more items.

In the definition above, the first parameter has been defined as a sequence of one or more items, the next two parameters as singleton-sequences. The function has been defined to count the number of items submitted by the first parameter, add the result to the sum of the two integers submitted by the other two parameters, and then divide the result by three to obtain the average. Notice the following:

- The first parameter is not enclosed in sequence-delimiter parentheses. (The parentheses around the first parameter are those of the `count()` function.) This is because the input parameter-value will submit the required single sequence, which is the single argument of the `count()` function.
- The sequence that is the argument of the `avg()` function is enclosed in parentheses. This is to specify that the argument of the `avg()` function is a single sequence consisting of three items that are the three integers submitted by the `count()` function and the two parameters `b` and `c`.

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 parameter-values, corresponding to the three parameters. The first parameter is enclosed in parentheses to delimit it as the first parameter-value. 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 parameter-values. The first parameter is allowed to be a sequence of one item. No parentheses are required to indicate separate input parameter-values.
- `sps:Average( (1 to 10),(20),30 )` returns `20`. There are three valid input parameter-values, corresponding to the three parameters. Parentheses may be used to indicate separate input parameter-values.
- `sps:Average( (10,20),(30),(40,50) )` returns an error because the third input parameter-value is not valid. It is a multi-item sequence, whereas the third parameter was defined as a singleton-sequence.
- `sps:Average( (10,20,30), 40, () )` returns an error because the third input parameter-value is an empty sequence. It must be a sequence with a single integer item.

Additional points of interest

The following additional points should be noted:

- If a 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(($a))`. This function would accept as its argument a single sequence of one-or-more items. The function could be called as follows: `sps:MyAverage((2,3,4))`, and it would return the value `3`. The input parameter-value must be enclosed in parentheses to ensure that the input is being read as a single sequence rather than as three singleton-sequences (which would be the case if there were to be no enclosing parentheses but commas separating the input values).
- If a parameter `$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 submitted as the first input parameter-value. 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 not count as an input value). The following, however, would generate an error: `sps:MyAverage(20,10)`, because the first input parameter-value is not returned.

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)`.

 © 2012 Altova GmbH

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 where it is located in the design. 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 parameter-values.

Consider the user-defined XPath function `Stock()`, which is defined with three parameters as in the screenshot below.

The definition in the function body is `$ContextNode[@name=$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 arguments of the function are its three parameter-values, each of which supplies either context or node-locator information. Alternatively, the following XPath expressions can be used as the function-call:

```
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. If a function is called from an XPath expression in a design component, then any parameter-value in the function-call is evaluated and it is the result of the evaluation that is passed as the parameter-value to the function.

Notice that in the function-call listed above there are three input parameter-values corresponding respectively to the three parameters:

- `$ContextNode = $XML/Trades/Stock` (the `/Trades/Stock` element)
- `$Selection = $XML/Trades/Selection/Stock` (the `/Trades/Selection/Stock` element)
- `$StockInfo = @name`

The XPath expression in the function definition is:

```
$ContextNode[@name=$Selection]/$StockInfo
```

When the input parameter-values 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. Functions can then locate the required nodes.

 © 2012 Altova GmbH

12.5 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 (*see examples below*). StyleVision supports the `xs:date` or `xs:dateTime` datatypes by providing a wide range of date formatting possibilities via the Input Formatting 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.

Date calculations with XPath 2.0

Data involving dates can be manipulated with XPath 2.0 expressions in Auto-Calculations. Given below are a few examples of what can be achieved with XPath 2.0 expressions.

- The XPath 2.0 functions `current-date()` and `current-dateTime()` can be used to obtain the current date and date-time, respectively.
- Dates can be subtracted. For example: `current-date() - DueDate` would return an `xdt:dayTimeDuration` value; for example, something like `P24D`, which indicates a positive difference of 24 days.
- Time units can be extracted from durations using XPath 2.0 functions. For example: `days-from-duration(xdt:dayTimeDuration('P24D'))` would return the integer `24`.

Here is an XPath 2.0 expression in an Auto-Calculation. It calculates a 4% annual interest on an overdue amount on a per-day basis and returns the sum of the principal amount and the accumulated interest:

```
if     (current-date() gt DueDate)
then   (round-half-to-even(InvoiceAmount +
          (InvoiceAmount*0.04 div 365 *
             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.

Formatting Dates

A date in an XML document is saved in the format specific to the datatype of its node. For example, the value of an `xs:date` node will have the format `YYYY-MM-DD[±HH:MM]`, while the value of an `xs:dateTime` node will have the format `YYYY-MM-DDTHH:MM:SS[±HH:MM]`. These formats are said to be the lexical representations of that data. By default, it is the lexical representation of the data that is displayed in Authentic View and the output. However, in the SPS, the Value Formatting feature can be used to display dates in alternative formats in Authentic View and, in some cases, optionally in the output.

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

- `xs:date`
- `xs:dateTime`
- `xs:duration`
- `xs:gYear`
- `xs:gYearMonth`
- `xs:gMonth`
- `xs:gMonthDay`
- `xs:gDay`

Using Value Formatting to format date nodes

To format dates alternatively to the lexical format of the date node, do the following:

1. Select the `contents` placeholder or input field of the node. Note that value formatting can only be applied to nodes created **as contents or an input field**.
2. In the Properties sidebar, select the `content` item, and then the *Content* group of properties. Now click the Edit button ... of the *Value Formatting* property. This pops up the Value Formatting dialog (*screenshot below*).

 © 2012 Altova GmbH

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

3. To define an alternative format, select the *Format* radio button.
4. You can now select a predefined date format from the drop-down list of the combo box (*screenshot below*), or define your own format in the input field of the combo box. See Value Formatting Syntax for details about the syntax to use when defining your own format.

Using Value Formatting to format Auto-Calculations

When Auto-Calculations evaluate to a value that is a lexical date format, Value Formatting can be used to format the display of the result. Do this as follows:

1. Select the Auto-Calculation in the design.
2. In the Properties sidebar, select the `content` item, and then the *AutoCalc* group of properties. Now click the Edit button ... of the *Value Formatting* property. This pops up the Value Formatting dialog (*screenshot below*).

By default, the *Unformatted* radio button is selected.

3. To define an alternative format, select the *Format* radio button.
4. In the Options for XML Schema value pane, in the *Datatype* combo box, select the `date` datatype to which the Auto-Calculation will evaluate. In the *Format* combo box, you can then select a predefined date format from the drop-down list (available options depend on the selected datatype), or define your own format in the input field of the combo box. See Value Formatting Syntax for details about the syntax to use when defining your own format.

Applying Value Formatting to the output

The Value Formatting that you define applies to Authentic View. Additionally, some Value Formatting definitions—not all—can also be applied to HTML 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.

 © 2012 Altova GmbH

12.6 Using Scripts

In StyleVision, you can define JavaScript functions for each SPS in a JavaScript editor (available as a tab in the Design View). The function definitions created in this way are stored in the header of the HTML document and can be called from within the body of the HTML document. Such functions are useful when:

- You wish to achieve a complex result using multiple script statements. In this case it is convenient to write all the required scripts, as separate functions, in one location (the header) and refer to the functions subsequently in the design document.
- You wish to use a particular script at multiple locations in the design document.

How to define functions in the JavaScript Editor is described in the sub-section Defining JavaScript Functions.

In the GUI, all JavaScript functions which are defined for a given SPS in the JavaScript Editor are listed in the Design Tree window under the Scripts entry (*screenshot below*). The screenshot below indicates that four JavaScript functions, `Average`, `ImageOut`, `ImageOver`, and `Buttons`, are currently defined in the active SPS.

The functions defined in the JavaScript Editor are available as event handler calls within the GUI. When a component in the design document is selected, any of the defined functions can be assigned to an event handler property in the Event property group in the Properties sidebar. How to assign a JavaScript function to an event handler is described in the section Assigning Function to Event Handlers.

Scripts in modular SPSs

When an SPS module is added to another SPS module, the scripts in the added module are available within the referring SPS, and can be used as event handlers via the Properties sidebar for components in the referring SPS. For more information about using modular SPSs, see the section Modular SPSs.

Defining JavaScript Functions

To define JavaScript functions, do the following:

1. In Design View, switch to the JavaScript Editor by clicking the Design View tab and selecting JavaScript (*screenshot below*).

2. In the JavaScript Editor, type in the function definitions (*see screenshot below*).

```
1   function DisplayTime()
2   {
3   now = new Date();
4   hours = now.getHours();
5   mins = now.getMinutes();
6   secs = now.getSeconds();
7   result = hours + ":" + mins + ":" + secs;
8   alert(result);
9   }
10
11  function ClearStatus()
12  {
13  window.status="";
14  }
```

The screenshot above shows the definitions of two JavaScript functions: `DisplayTime` and `ClearStatus`. These have been described for the active SPS. They will be entered in the header of the HTML file as follows:

```
<script language="javascript">

<!-- function DisplayTime()
{
    now = new Date();
    hours = now.getHours();
    mins = now.getMinutes();
    secs = now.getSeconds();
    result = hours + "." + mins + "." + secs;
    alert(result)
}

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

</script>
```

These functions can now be called from anywhere in the HTML document. In StyleVision, all the defined functions are available as options that can be assigned to an event handler property in the *Event* property group in the Properties sidebar. See Assigning Function to Event Handlers for details.

 © 2012 Altova GmbH

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.

External JavaScript Files

An SPS can access external JavaScript files in two ways:

1. By creating a User-Defined Element or User-Defined XML Block. These design objects can contain a SCRIPT element that accesses the external JavaScript file. Note that location of the User-Defined Element or User-Defined XML Block is within the BODY element of the design (and therefore within the BODY element of the HTML output, not within the HEAD element).
2. By adding a script in the Javascript Editor that accesses the external file. A script that is added in this way will be located in the HEAD element of the HTML output.

User-Defined Elements and User-Defined XML Blocks

External JavaScript files can be accessed by means of User-Defined Elements and User-Defined XML Blocks. Using these mechanisms, a SCRIPT element that accesses the external JavaScript file can be inserted at any location within the BODY element of the output HTML document.

A User-Defined Element could be inserted as follows:

1. Place the cursor at the location in the design where the SCRIPT element that accesses the JavaScript file is to be inserted.
2. From the **Insert** menu or context menu, select the command for inserting a User-Defined Element.

3. In the dialog that pops up (*see screenshot above*), enter the SCRIPT element as shown above, giving the URL of the JavaScript file as the value of the src attribute of the SCRIPT element: for example, `script type="text/javascript" src="file:///c:/Users/mam/Desktop/test.js"`
4. Click **OK** to finish.

You can also use a User-Defined XML Block to achieve the same result. To do this use the same procedure as described above for User-Defined Elements, with the only differences being (i) that a User-Defined XML Block is inserted instead of a User-Defined Element, and (ii) that the SCRIPT element is inserted as a complete XML block, that is, with start and end tags.

 © 2012 Altova GmbH

JavaScript Editor

The JavaScript Editor enables you to insert an external script in the `HEAD` element of the HTML output. Do this by entering, in the JavaScript Editor, the following script fragment, outside any other function definitions that you create.

```
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.

12.7 HTML Import

In StyleVision you can import an HTML file and create the following documents based on it:

- An SPS document based on the design and structure of the imported HTML file.
- An XML Schema, in which HTML document components are created as schema elements or attributes. Optionally, additional elements and attributes that are not related to the HTML document can be created in the user-defined schema.
- An XML document with: (i) a structure based on the XML Schema you have created, and (ii) content from the HTML file.
- XSLT stylesheets based on the design in Design View.

HTML-to-XML: step-by-step

The HTML Import mechanism, which enables the creation of XML files based on the imported HTML file, consists of the following steps:

1. Creating New SPS via HTML Import. When an HTML file is imported into StyleVision, a new SPS document is created. The HTML document is displayed in Design View with HTML markup tags. A user-defined XML Schema with a document element called `Root` is created in the Schema Tree window. This is the schema on which the SPS is based. The HTML document content and markup that is displayed in Design View at this point is included in the SPS as static content.
2. Creating the Schema and SPS Design. Create the schema by (i) dragging components from the HTML document to the required location in the schema tree (in the Schema Tree window); and, optionally, (ii) adding your own nodes to the schema tree. In the Design Window, HTML content that has been used to build nodes in the schema tree will now be displayed with schema node tags around the content. HTML content that has no corresponding schema node will continue to be displayed without schema node tags.
3. In the Design Document, assign formatting to nodes, refine processing rules, or add static content as required. These modifications will have an effect only on the SPS and the generated XSLT. It will not have an effect on either the generated schema or XML file.
4. After you have completed the schema tree and the design of the SPS, you can generate and save the following:
   - an XML Schema corresponding to the schema tree you have created;
   - an XML data file with a structure based on the schema and content for schema nodes that are created with the `(content)` placeholder in the SPS design;
   - a SPS (`.sps` file) and/or XSLT stylesheet based on your design.

 © 2012 Altova GmbH

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 and click **OK**.

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 `Root`.

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).

 © 2012 Altova GmbH

Creating the Schema and SPS Design

The schema is created by dragging selections from Design View into the user-defined schema. You do this one selection at a time. The selection is dropped on a node in the schema tree (relative to which the new node will be created, either as a child or sibling). You select the type of the node to be created (element or attribute) and whether the selection is to be converted to the new node or surrounded by it.

The selection

The selection in Design View can be any of the following:

- A node in the HTML document.
- A text string within a node.
- Adjacent text strings across nodes.
- An image.
- A link.
- A table.
- A list.
- A combination of any of the above.

In this section we explain the process in general for any selection. The special cases of tables and lists are discussed in more detail in the section Creating Tables and Lists as Elements/ Attributes.

To make a selection, click an HTML document component or highlight the required text string. If multiple components are to be selected, click and drag over the desired components to highlight the selection. Note that StyleVision extends the selection at the beginning and end of the selection to select higher-level elements till the first and last selected elements belong to the same parent.

The location in the schema tree

On dragging the selection over the desired schema tree node, one of the following symbols will appear together with the popup message: `Create new schema item`.

- 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. In the schema tree (in the Schema Tree sidebar), such an XML node is indicated by parentheses containing an ellipsis.
- The inserted node tags are inserted with the necessary path (that is, with ancestor node tags that establish a path relative to the containing node). The path will be absolute or relative depending on the context of the node in the design.
- How to create nodes from table and list selections are described in Creating Tables and Lists as Elements/Attributes.

Adding and deleting nodes in the schema
You can add additional nodes (which are not based on an HTML selection) to the user-defined schema. Do this by right-clicking on a node and selecting the required command from the context menu. Alternatively, you can use the toolbar icons of the Schema Tree sidebar.

To delete a node, select the node and then use either the context menu or the toolbar icon. Note, however, that when a node is deleted, some paths in the design could be invalidated.

Modifying the design
You can modify the structure of the design by dragging components around and by inserting static and dynamic components. Styles can also be modified using the various styling capabilities of StyleVision.

 © 2012 Altova GmbH

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 table created in the SPS should be a static table or dynamic table.

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.

 © 2012 Altova GmbH

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.

12.8 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:

- 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.)
- Also from the SPS, generate the XSLT-for-HTML file.
- Transform the XML file using the generated XSLT-for-HTML file.
- 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 AltovaXML application installed in order for the XSLT transformation to run correctly.

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`

 © 2012 Altova GmbH

`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 AltovaXML, together with the `SPSFilename.xsltt` file. (AltovaXML is Altova's XSLT transformation engine. It can be downloaded from the Altova website.) AltovaXML 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.

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 AltovaXML

Make sure that the latest version of AltovaXML is installed. AltovaXML is Altova's transformation engine; it will be used to transform the DB-generated XML file.

Activate Internet Information Server (Microsoft's web server)

If Internet Information Server (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. Make sure that the database and the SPS file are 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.

In StyleVision, generate the XSLT-for-HTML file

Generate the XSLT-for-HTML file by issuing the command *Files* | *Save Generated Files* | *Save generated XSLT-HTML file*. Save the file to the same folder as that in which the ASPX application is.

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*, select the command **Advanced Settings**.
6. For the *Identity* property, select *Custom account* and enter your Windows user name

 © 2012 Altova GmbH

and password.

7. For the *Enable 32-Bit Applications* property, enter `True`. (This is so the database drivers have access).

Run the application

In the browser, go to `localhost/Test1/Test.aspx` (assuming `Test1` is the name of the folder in which the ASPX application has been saved).

Now the transformed HTML should be visible. If the browser hangs at this point, make sure the AltovaXML license file is in place. Refreshing this page will cause the latest data from the database to be displayed.

Chapter 13

Automated Processing

13 Automated Processing

The functionality of StyleVision together with the various XSLT and output files generated by StyleVision provide powerful automation possibilities. This section describes these capabilities.

StyleVision's file-generation functionality
After you have created an SPS design with StyleVision, you can generate several kinds of XSLT and output files from within the GUI, depending on which edition of StyleVision you are using (Enterprise, Professional, or Basic). The following files can be generated with the **File | Save Generated Files** 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.

StyleVisionBatch and AltovaXML: generating files from outside the GUI
Additionally to generating XSLT stylesheets and the required output formats via the StyleVision GUI (**File | Save Generated Files** command), you can generate output files using two other methods:

1. With the StyleVisionBatch utility, which calls StyleVision's file generation functionality without opening the GUI, You can use various input parameters to produce various kinds of output. One parameter you can specify is the SPS file itself, from which all XSLT stylesheets can be generated, and hence all end output formats. (The Enterprise and Professional Editions offer multiple output formats.) StyleVisionBatch is used from the command line and thus enables the automation of StyleVision's file-generation functionality. How to use StyleVisionBatch is explained in the sub-section, Command Line Interface: StyleVisionBatch.
2. With AltovaXML, a free, standalone Altova application that contains the Altova XML Validator, Altova XSLT Engines (1.0 and 2.0), and Altova XQuery 1.0 Engine. The XSLT Engines in AltovaXML can be used for transformations of XML to an output format by processing XML documents with XSLT stylesheets. As a result, the XSLT file will have to be created in advance so that it can be provided as an input parameter to AltovaXML. (AltovaXML does not take an SPS as an input parameter.) The advantages of using AltovaXML are: (i) the savings on time and memory overheads compared to using StyleVisionBatch; and (ii) in addition to a command line interface, AltovaXML provides interfaces for COM, Java, and .NET, and can therefore be easily called from within these environments. How to use AltovaXML for transformations is explained in the sub-section AltovaXML.

Automation with scheduled tasks
Since both StyleVisionBatch and AltovaXML can be called from the command line, their functionality can be automated and scheduled. How to do this is explained in the section, How to Automate Processing.

 © 2012 Altova GmbH

13.1 Command Line Interface: StyleVisionBatch

StyleVision's file-generation functionality can be called via the StyleVisionBatch utility, which is included in your StyleVision installation. The utility is named `StyleVisionBatch.exe` and is located in the StyleVision application folder. The syntax for invoking StyleVision commands via StyleVisionBatch is explained in the StyleVisionBatch Syntax sub-section. When a command is executed StyleVision runs silently (i.e. without the GUI being opened), generates the required output files, and closes.

Output files
Using StyleVisionBatch, you can generate one or more of the following files:

- XSLT-for-HTML (`.xslt`) file from the specified SPS
- HTML (`.html`) file using the XML and XSLT files in the specified SPS or using alternative XML and/or XSLT files

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 DOS window you can go to the directory in which the StyleVisionBatch utility is, then enter a command such as: `StyleVisionBatch -v Test.sps -OutXSLT=Test.xslt.`, 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
StyleVisionBatch -v Test.sps -inpXSLT=EN.xslt -OutHTML=TestEN.html
StyleVisionBatch -v Test.sps -inpXSLT=DE.xslt -OutHTML=TestDE.html
StyleVisionBatch -v Test.sps -inpXSLT=FR.xslt -OutHTML=TestFR.html
```

  When the batch file is processed, the commands are executed and the files generated.

StyleVision functionality in scheduled tasks
Using the Scheduled Tasks tool of Windows, StyleVisionBatch 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 StyleVisionBatch commands as a scheduled task is described in How to Automate Processing.

StyleVisionBatch Syntax

The syntax for the command line interface utility StyleVisionBatch is:

```
StyleVisionBatch [ <Stylevision exe>] [ <options>]
```

where

`StyleVisionBatch`	is the CLI utility, which is located in the StyleVision application folder
`<Stylevision exe>`	is the StyleVision executable file; it needs to be specified only if the StyleVision executable is **not** named `stylevision.exe` **or** is not located in the same folder as `StyleVisionBatch.exe`. If specified, the name must end in `.exe`.
`<options>`	One or more of the options listed below.

StyleVisionBatch options
StyleVisionBatch options may be entered in any order. In the listing below they are organized into groups so as to provide a better overview. 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.

- ***Utility***

`-help` or `-?`	Displays syntax at the command line
`-verbose` or `-v`	Displays processing information at runtime
`-FOPBatFile=<file>`	Sets FOP processor batch file *(Enterprise edition)*

- ***SPS and Parameters***

`<stylesheet>`	Sets SPS (`.sps`) stylesheet
`$<paramname>=<value>`	Assigns a value to a stylesheet parameter. If the value contains a space, enclose the value in double quotes. For example: `$paramname="A value"`. Multiple parameters are separated by spaces.

- ***XSLT file output***

`-OutXSLT=<file>`	Writes XSLT-for-HTML to the specified file
`-OutXSLRTF=<file>`	Writes XSLT-for-RTF to the specified file *(Enterprise and Professional editions)*
`-OutXSLFO=<file>`	Writes XSLT-for-FO to the specified file *(Enterprise edition only)*
`-OutXSLWord2007=<file>`	Writes XSLT-for-Word 2007+ to the specified file *(Enterprise edition only)*

- ***Input files***

 © 2012 Altova GmbH

-InpXML=<file>	Sets input XML file
-InpXSLT=<file>	Sets input XSLT-for-HTML file
-InpXSLRTF=<file>	Sets input XSLT-for-RTF file
-InpXSLFO=<file>	Sets input XSLT-for-FO file *(Enterprise edition)*
-InpXSLWord2007=<file>	Sets input XSLT-for-Word 2007+ file *(Enterprise edition)*

- ***Output files***

-OutHTML=<file>	Writes HTML output to the specified file
-OutRTF=<file>	Writes RTF output to the specified file *(Enterprise and Professional editions)*
-OutFO=<file>	Writes FO output to the specified file *(Enterprise edition)*
-OutPDF=<file>	Writes PDF output to the specified file *(Enterprise edition)*
-OutWord2007=<file>	Writes Word 2007+ output to the specified file *(Enterprise edition)*

Explanatory points

The following points provide supplementary information about StyleVisionBatch syntax and the command line process.

- When StyleVisionBatch is called, it looks in the current directory for StyleVision.exe. If your StyleVision executable is named otherwise or located in another folder, use the <Stylevision exe> argument to specify the executable.
- Paths may be absolute or relative and should use backslashes.
- Options are prefixed either with a minus sign (for example: -OutHTML) or a forward slash (for example: /OutHTML).
- 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.
- If the SPS file is specified, the Working XML File associated with it and the XSLT stylesheet generated from it will be used to generate output; therefore no input XML or XSLT file is required. If, however, the SPS file is not specified, an input XML file and input XSLT file must be specified as options. An input XML File must also be specified if the SPS file does not have a Working XML File assigned to it.
- Parameter declarations refer to parameters in the XSLT stylesheet. Parameter names and values are case-sensitive. Each parameter declaration on the command line must be prefixed with a $, and, if multiple parameters are used, they must be separated from each other with a space. If the value of the parameter contains a space, then the value must be enclosed in double quotes.
- No default output is specified, so you must specify the required output. For example: OutHTML=Test.html.
- If you specify only the output file (no XML file or XSLT file), the Working XML File specified in the SPS is used for the source XML, and the required XSLT is generated from the SPS.
- Any temporary files that are created are deleted at the end of the processing.
- The -verbose option provides a detailed report of all steps carried out during the processing of the command.
- When specifying HTML output, make sure that the generated file is placed in a location

in which relative paths to images, etc, will point correctly to their targets. The same applies to hyperlinks.

© 2012 Altova GmbH

StyleVisionBatch Examples

The examples below are organized according to output.

XSLT stylesheets

XSLT stylesheets can be generated from the SPS files. The only input required is the SPS file.

- The XSLT-for-HTML file is generated from the SPS.
```
StyleVisionBatch -v Test.sps -OutXSLT=Test.xslt
```

HTML output

HTML output is obtained by transforming an XML file with an XSLT stylesheet. The XML file may be the Working XML File assigned in the SPS or may be specified on the command line. The XSLT file may be that generated from the SPS or may be specified on the command line.

- Working XML file in SPS transformed with XSLT stylesheet generated from SPS.
```
StyleVisionBatch -v Test.sps -OutHTML=Test.html
```
- Specified XML file transformed with XSLT stylesheet generated from SPS.
```
StyleVisionBatch -v Test.sps -InpXML=External.xml -OutHTML=Test.html
```
- Working XML file in SPS transformed with specified XSLT stylesheet.
```
StyleVisionBatch -v Test.sps -InpXSLT=External.xslt -OutHTML=Test.html
```
- Specified XML file transformed with specified XSLT stylesheet.
```
StyleVisionBatch -v -InpXML=External.xml  -InpXSLT=External.xslt
-OutHTML=Test.html
StyleVisionBatch -v Test.sps -InpXML=External.xml
-InpXSLT=External.xslt -OutHTML=Test.html
```

Parameter Usage

For the XSLT transformation, parameters can be passed to the XSLT stylesheet from the command line.

- Parameters passed to XSLT stylesheet generated from the SPS.
```
StyleVisionBatch -v Test.sps -OutHTML=Test.html $myparam=MyText
StyleVisionBatch -v Test.sps -inpXML=External.xml -OutHTML=Test.html
$myparam="My Text"
StyleVisionBatch -v Test.sps -OutHTML=Test.html -OutFO=Test.fo
$myparam="MyText"
StyleVisionBatch -v Test.sps -OutHTML=Test.html $myparam=2006
StyleVisionBatch -v Test.sps -OutHTML=Test.html $myparam="2006"
```
- Parameters passed to specified XSLT stylesheet.
```
StyleVisionBatch -v Test.sps -inpXSLT=External.xslt -OutHTML=Test.html
$myparam=MyText
StyleVisionBatch -v Test.sps -inpXSLT=External.xslt -OutHTML=Test.html
$myparam="My Text"
```

13.2 Using AltovaXML

AltovaXML is a free product that contains the Altova XML Validator, XSLT 1.0 and 2.0 Engines, and XQuery 1.0 Engine. It is downloadable from the Altova website. AltovaXML can be run from the command line and has interfaces for COM, Java, and .NET. You can therefore easily use AltovaXML from within these environments to validate XML documents, perform XSLT transformations, and execute XQuery documents.

The functionality of AltovaXML that would be most relevant to StyleVision users is the XSLT 1.0 and 2.0 transformation functionality. Typically, this functionality would be used as follows:

1. An XSLT stylesheet is generated from an SPS with the **File | Save Generated Files** command or by using StyleVisionBatch. Note that AltovaXML cannot be used to generate XSLT stylesheets from an SPS file in the way that StyleVisionBatch does.
2. The generated XSLT stylesheet is used to transform XML documents with AltovaXML.

With AltovaXML you can generate HTML output.

Advantages of AltovaXML
The advantages of using AltovaXML are as follows:

- AltovaXML is a leaner package than StyleVisionBatch and therefore provides faster validation and XSLT transformation. This is because StyleVisionBatch uses the Altova Validator and XSLT Engines in StyleVision, and requires more memory and time overhead as a result.
- Easy use with command line, COM, Java, and .NET interfaces.
- Automation and scheduling with the use of batch files and the scheduling processes such as the Scheduled Tasks process of Windows.

In this section
This section is organized into the following sub-sections:

- XSLT 1.0 CLI Transformations describes the syntax for calls to the Altova XSLT 1.0 Engine of AltovaXML and provides examples of use.
- XSLT 2.0 CLI Transformations describes the syntax for calls to the Altova XSLT 2.0 Engine of AltovaXML and provides examples of use.

For a description of how AltovaXML can be used to automate the production of output documents (such as HTML) from XML source documents, see the section How to Automate Processing.

For additional and more detailed information about using AltovaXML, including how to use AltovaXML's COM, Java, and .NET interfaces, see the AltovaXML user documentation.

 © 2012 Altova GmbH

XSLT 1.0 CLI Transformations

Syntax

The syntax to invoke XSLT 1.0 transformations is:

AltovaXML -xslt1 *xsltfile* -in *xmlfile* [-out *outputfile*] [options]

where

`AltovaXML`	Calls the application.
`-xslt1`	Specifies that the Altova XSLT 1.0 Engine is to be used for an XSLT transformation; the engine uses the XSLT 1.0 file *`xsltfile`* for the transformation.
`-in`	Specifies the XML file `xmlfile` to be transformed and its location.
`-out`	Specifies the output file `outputfile` and its location. If this option is omitted, the output is written to standard output.

The following options are available:

`-param`	Takes the instruction `paramname=XPath expression`. The `-param` switch is used before each global parameter. Double quotes must be used if a space is included in an XPath expression—whether in a path expression itself or in a string literal in the expression. See examples.
`-xslstack`	The stack size is the maximum depth of executed instructions, and can be changed with the `-xslstack` value. The minimum allowed value is 100. The default stack size is 1000. If the stack size is exceeded during a transformation, an error is reported.
`-namedTemplate` (or `-n`)	Sets the initial named template. A space separates the argument from its value. Example: `-namedTemplate MyTemplate`
`-mode` (or `-m`)	Sets the initial template mode. A space separates the argument from its value. Example: `-mode MyMode`

Note:

- The XSLT file must be specified in the command line instruction; an XSLT file referenced in an `<?xml-stylesheet?>` processing instruction in the XML document is not automatically used.
- If the `-out` parameter is omitted, output is written to the standard output.

Examples

- `AltovaXML -xslt1 test.xslt -in test.xml -out testout.xml`
- `AltovaXML -xslt1 test.xslt -in test.xml -out testout.xml -param date=//node/@att1`
- `AltovaXML -xslt1 test.xslt -in test.xml -out testout.xml -param date="//node/@att1 | //node/@att2"`
- `AltovaXML -xslt1 test.xslt -in test.xml -out testout.xml -param date=node/@att1 -param title='stringwithoutspace'`
- `AltovaXML -xslt1 test.xslt -in test.xml -out testout.xml -param date=node/@att1 -param title="'string with spaces'"`

XSLT 2.0 CLI Transformations

Syntax
The syntax to invoke XSLT 2.0 transformations is:

AltovaXML -xslt2 *xsltfile* -in *xmlfile* [-out *outputfile*] [options]

where

`AltovaXML`	Calls the application.
`-xslt2`	Specifies that the Altova XSLT 2.0 Engine is to be used for an XSLT transformation; the engine uses the XSLT 2.0 file *`xsltfile`* for the transformation.
`-in`	Specifies the XML file `xmlfile` to be transformed and its location.
`-out`	Specifies the output file `outputfile` and its location. If this option is omitted, the output is written to standard output.

The following options are available:

`-param`	Takes the instruction `paramname=XPath expression`. The `-param` switch is used before each global parameter. Double quotes must be used if a space is included in an XPath expression—whether in a path expression itself or in a string literal in the expression. See examples.
`-xslstack`	The stack size is the maximum depth of executed instructions, and can be changed with the `-xslstack` value. The minimum allowed value is 100. The default stack size is 1000. If the stack size is exceeded during a transformation, an error is reported.
`-namedTemplate` (or `-n`)	Sets the initial named template. A space separates the argument from its value. Example: `-namedTemplate MyTemplate`
`-mode` (or `-m`)	Sets the initial template mode. A space separates the argument from its value. Example: `-mode MyMode`

Note:

- The XSLT file must be specified in the command line instruction; an XSLT file referenced in an `<?xml-stylesheet?>` processing instruction in the XML document is not automatically used.
- If the `-out` parameter is omitted, output is written to the standard output.
- The XSLT 2.0 Engine can be used in its backward compatibility mode to process an XSLT 1.0 stylesheet. The output, however, could be different than that produced by the XSLT 1.0 Engine processing the same XSLT 1.0 stylesheet.

Examples

- `AltovaXML -xslt2 test.xslt -in test.xml -out testout.xml`
- `AltovaXML -xslt2 test.xslt -in test.xml -out testout.xml -param date=//node/@att1`
- `AltovaXML -xslt2 test.xslt -in test.xml -out testout.xml -param date="//node/@att1 | //node/@att2"`
- `AltovaXML -xslt2 test.xslt -in test.xml -out testout.xml -param date=node/@att1 -param title='stringwithoutspace'`
- `AltovaXML -xslt2 test.xslt -in test.xml -out testout.xml -param date=node/@att1 -param title="'string with spaces'"`

 © 2012 Altova GmbH

13.3 How to Automate Processing

Processing can be automated in two ways:

- Commands can be specified to execute one after another. This automates the execution of a sequence of commands. Such automation is easily achieved by means of batch files, and is described in the sub-section, Creating Batch Files.
- A command or a set of commands can be specified to execute at a given time. This is achieved through the Scheduled Tasks tool of Windows, described in the sub-section Automating with Scheduled Tasks.

Creating Batch Files

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.) *See below*.
- Scheduling a sequence of tasks to be executed at a particular time. See Automating with Scheduled Tasks.

Batch file with sequence of commands

A sequence of commands to be executed is entered as follows:

```
@ECHO OFF
CLS
StyleVisionBatch -v Test.sps -inpXSLT=EN.xslt -OutHTML=TestEN.html
StyleVisionBatch -v Test.sps -inpXSLT=DE.xslt -OutHTML=TestDE.html
StyleVisionBatch -v Test.sps -inpXSLT=FR.xslt -OutHTML=TestFR.html
```

When the batch file is processed, the commands are executed and the files generated. The batch file above uses StyleVisionBatch to generate three HTML outputs, each being generated with a different XSLT stylesheet. The input file is the Working XML File of the SPS file.

Batch file that uses output from preceding command

This batch file calls AltovaXML to generate an XML file and then uses this XML file as input for an XSLT transformation with StyleVisionBatch. (For the sake of simplicity, assume that the calls to AltovaXML and StyleVisionBatch correctly locate the executables.)

```
@ECHO OFF
CLS
AltovaXML -xslt2 Test.xslt -in Test.xml -out TestOut.xml
StyleVisionBatch -v Test.sps -inpXML=TestOut.xml -OutHTML=TestOut.html
```

When the batch file is processed, `TestOut.xml` is generated by the first command. The second command takes `TestOut.xml` as its input XML file and processes it with the XSLT-for-HTML stylesheet generated on the fly by the SPS file `Test.sps`. The output is the HTML file `TestOut.html`.

 © 2012 Altova GmbH

Automating with Scheduled Tasks (Windows XP)

A command or set of commands (that call StyleVisionBatch or AltovaXML, for example) can be set up to run to a pre-determined schedule. This scheduling is done with the Scheduled Tasks tool of Windows. The Scheduled Task tool opens the utility or application called and executes the command specified in the task.

To create a scheduled task, do the following. The example below uses StyleVisionBatch as the program to call; to use AltovaXML, simply substitute AltovaXML for StyleVisionBatch.

1. If you plan to run a *set* of StyleVisionBatch commands as a scheduled task—as opposed to a single command—these commands should be created in a batch file (*see Creating Batch Files*) and the batch file should be specified as the command to execute. If a single StyleVisionBatch command is to be scheduled, skip Step 1 (this step) and go to Step 2.
2. Open the Scheduled Task Wizard of Windows (**Start | Control Panel | Scheduled Tasks | Add Scheduled Task**).
3. Click **Next** to start setting up the task.
4. In the window to select the program to run, you select either `StyleVisionBatch.exe` (for a single StyleVisionBatch command) or a batch file (containing multiple StyleVisionBatch commands). Browse for the required file and select it. The next screen (*screenshot below*) appears.

5. Assign a name for the task, and set a frequency for it. Then click **Next**.
6. Select the starting day and time for the schedule. Then click **Next**.
7. Enter the appropriate user name and password. Then click **Next**.
8. In the finishing screen (*screenshot below*), if you are scheduling a single StyleVisionBatch command and have therefore selected `SVBATCH%>.exe` as the program to run, check the Open Advanced Properties... check box. (It is in the Advanced Properties dialog that the StyleVisionBatch command is specified.) Then click **Finish**.

If you have specified a batch file as the program to run for the task, there is no need to set any advanced properties and you can leave the Open Advanced Properties check box unchecked. In this case, the scheduling of the task is now complete.

9. This step is required only if you are scheduling a single StyleVisionBatch command as your task. On clicking **Finish** with the Open Advanced Properties... check box checked, a dialog showing the properties of the task pops up (*screenshot below*).

 © 2012 Altova GmbH

In the Start In text field (*screenshot above*) enter the required StyleVisionBatch command, for example: `"C:\Program Files\Altova\StyleVision20XX" -v Examples\NanonullOrg.sps -OutXSLT=Examples\Nano1.xslt`. Use quotes if there are spaces in your file or folder names, and, in your paths, use backslashes. If desired, enter a comment describing the task. Click **OK** to finish.

Deleting a scheduled task

To delete a scheduled task, open the Scheduled Tasks window (**Start | Control Panel | Scheduled Tasks**), select the task and either click the **Delete** icon or press the **Delete** key.

Automating with Scheduled Tasks (Windows Vista)

A command or set of commands (that call StyleVisionBatch or AltovaXML, for example) can be set up to run to a pre-determined schedule. This scheduling is done with the Scheduled Tasks tool of Windows. The Scheduled Task tool opens the utility or application called and executes the command specified in the task.

To create a scheduled task on a Windows Vista machine, do the following. The example below uses StyleVisionBatch as the program to call; to use AltovaXML, simply substitute AltovaXML for StyleVisionBatch.

1. If you plan to run a *set* of StyleVisionBatch commands as a scheduled task—as opposed to a single command—these commands should be created in a batch file (*see* *Creating Batch Files*) and the batch file should be specified as the command to execute. If a single StyleVisionBatch command is to be scheduled, skip Step 1 (this step) and go to Step 2.
2. Select **Start | Settings | Control Panel**).
3. Double-click **Administrative Tools** (*see screenshot below*).

4. In the Administrative Tools window, double-click **Task Scheduler** (*screenshot below*).

 © 2012 Altova GmbH

5. The Task Scheduler window appears (*screenshot below*). Double-click **Create Basic Task**.

6. In the Create Basic Task window (*screenshot below*), select **Trigger** and set the trigger as required. In the screenshot below, the trigger has been set to *One time*. Then click **Next**.

7. Select One Time in the menu bar (*screenshot below*) and set the time. Then click **Next**.

8. Select Action in the menu bar (*screenshot below*) and click *Start a program*. Then click **Next**.

9. Select *Start a program* in the menu bar (*screenshot below*) and select either `StyleVisionBatch.exe` (for a single StyleVisionBatch command) or a batch file (containing multiple StyleVisionBatch commands). Browse for the required file and select it.

If you are using StyleVisionBatch (and not a batch file), then, in the *Start In* text field (*screenshot above*), enter the required StyleVisionBatch command, for example: `"C:\Program Files\Altova\StyleVision20XX" -v Examples\NanonullOrg.sps -OutXSLT=Examples\Nano1.xslt`. Use quotes if there are spaces in your file or folder names, and, in your paths, use backslashes. If you are using a batch file, the *Start In* text field should be left empty. Then click **Next**.

10. In the Finish window (*screenshot below*), assign a name for the task and create a description of it. If you are scheduling a single StyleVisionBatch command and have therefore selected `SVBATCH%>.exe` as the program to run, check the *Open the Properties dialog...* check box. Then click **Finish**.

If you have specified a batch file as the program to run for the task, you can leave the *Open the Properties* check box unchecked. In this case, the scheduling of the task is now complete.

11. This step is required only if you are scheduling a single StyleVisionBatch command as your task. On clicking **Finish** with the Open Properties check box checked, a dialog showing the properties of the task pops up. Check that the command you have specified for StyleVisionBatch to execute is correct.

 © 2012 Altova GmbH

Chapter 14

Reference

14 Reference

This section contains a complete description of StyleVision toolbars, Design View symbols, and menu commands. It is divided into the following broad parts:

- A description of all the toolbars with their icons, as well as a description of how to customize the views of the toolbars.
- Descriptions of symbols used in Design View and of the Edit XPath Expression dialog.
- All menu commands.

While the User Reference section contains a description of individual commands, the mechanisms behind various StyleVision features are explained in detail in the relevant sections. The mechanisms have been organized into the following groups::

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

For command line usage, see Command Line Interface: StyleVisionBatch.

 © 2012 Altova GmbH

14.1 Toolbars

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

- Formatting
- Table
- Insert Design Elements
- Design Filter
- Global Resources
- Standard

The icons in each toolbar are listed in the sub-sections of this section, each with a brief description of the corresponding command.

Positioning the toolbars
A toolbar can float freely on the screen or can be placed in a toolbar area along any edge of the GUI. Toolbars are most commonly placed along the top edge of the GUI, just below the Menu bar. However, they can also be placed along the side or bottom edges of the GUI.

To position a toolbar in a toolbar area, do the following:

1. Grab the toolbar by its handle (if the toolbar is already in a toolbar area) or by its title bar (if the toolbar is floating).
2. Drag the toolbar to the desired toolbar area, if it exists, and drop it at the desired location in that toolbar area. If no toolbar area exists at the edge along which you wish to place the toolbar, dragging the toolbar to that edge will automatically create a toolbar area there when the toolbar is dropped.

To make a toolbar float freely grab it by its handle, drag it away from the toolbar area, and drop it anywhere on the screen except at an edge or in an existing toolbar area.

Switching the display of toolbars on and off
The display of individual toolbars can be switched on and off using any of the following three methods:

- In the **View | Toolbars** menu (*screenshot below*), select or deselect a toolbar to, respectively, show or hide that toolbar.

- Right-click any toolbar area to display a context menu (*screenshot below*) that allows you to toggle the display of individual toolbars on and off.

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

Adding and removing toolbar buttons

Individual toolbar buttons can be added to or removed from a toolbar, that is, they can be made visible or be hidden. To add or remove a button from a toolbar, do the following:

1. In the toolbar where the button to be added or removed is, click the **More Buttons** button (if the toolbar is in a toolbar area) or the **Toolbar Options** button (if the toolbar is a floating toolbar). The **More Buttons** button is an arrowhead located at the right-hand side of the toolbar (in horizontal toolbar areas) or at the bottom of the toolbar (in vertical toolbar areas). The Toolbar Options button is an arrowhead located at the right-hand side of the floating toolbar.
2. In the **Add or Remove Buttons** menu that pops up, place the cursor over the **Add or Remove Buttons** menu item (*screenshot below*). This rolls out a menu which contains the names of the toolbars in that toolbar area plus the **Customize** menu item (*screenshot below*).

 © 2012 Altova GmbH

3. Place the cursor over the toolbar that contains the toolbar button to be added or removed (*screenshot above*).
4. In the menu that rolls out (*screenshot above*), click on the name of the toolbar button to add or remove that button from the toolbar.
5. Clicking the Customize item pops up the Customize dialog.

The **Reset Toolbar** item below the list of buttons in each toolbar menu resets the toolbar to the state it was in when you downloaded StyleVision. In this state, all buttons for that toolbar are displayed.

Note: The buttons that a toolbar contains are preset and cannot be disassociated from that toolbar. The process described above displays or hides the button in the toolbar that is displayed in the GUI.

Formatting

The **Formatting toolbar** (*screenshot below*) contains commands that assign commonly used inline and block formatting properties to the item/s selected in the SPS.

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 `<div>` element around the current selection in Design View.

Text properties
The bold, italic, and underline inline text properties can be directly applied to the current selection in Design View by clicking on the appropriate button.

Alignment
Alignment properties (left-aligned, centered, right-aligned, and justified) can be directly applied to the selection in Design View.

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

Hyperlinks
Inserts a hyperlink at the cursor insertion point. See Hyperlink for a description of how to use this command.

 © 2012 Altova GmbH

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).

Row and Column operations
Rows and columns in any SPS table (static or dynamic) can be inserted, appended, or deleted with reference to the cursor location. Rows and columns are inserted before the current cursor location or appended after all rows/columns. The row/column in which the cursor is can also be deleted. These operations are achieved with the **Insert Row/Column**, **Append Row/Column**, or **Delete Row/Column** buttons. You can also add table headers and footers as either columns or rows **Add Table Header/Footer Column/Row**.

Cell operations
An SPS table cell in which the cursor is located can be joined to any one of the four cells around it. The joining operation is similar to that of spanning table cells in HTML. The buttons to be used for these operations are **Join Cell Right/Left/Above/Below**. Also, an SPS table cell in which the cursor is located can be split, either horizontally or vertically, using the **Split Cell Horizontally** and **Split Cell Vertically** buttons, respectively. SPS table cell content can be aligned vertically at the top, in the middle, and at the bottom. The display of cell borders can be switched on and off with the **View Cell Bounds** toggle.

Table operations, properties, display
Placing the cursor in a static or dynamic table and clicking **Delete Table** deletes that table. Table markup can be toggled on and off with the View Table Markup command. The Table Properties command pops up the Table Properties dialog, in which properties of the table can be defined.

 © 2012 Altova GmbH

Insert Design Elements

The **Insert Design Elements toolbar** contains icons for commands to insert design elements in the SPS design, and for related commands. The various design elements that can be inserted via these toolbar icons are shown in the screenshot below. There are three types of items in the toolbar:

1. Design elements, which are context-node-sensitive (the majority of elements in the toolbar),
2. Layout elements, which are independent of node context, and
3. Grid-related toggles to aid design.

Design elements

The design elements are the context-node-sensitive elements that are available in the **Insert** menu. To insert a design element using its toolbar icon, do the following:

1. Select the toolbar icon for the element you wish to insert.
2. Click the location in the design where the element is to be inserted. A Insert Design

Element for the selected design element (*screenshot below*) pops up. This displays the schema tree with the context node highlighted. The context node is the node within which the cursor has been placed for the insertion of the design element.

3. If you wish to insert the design element within the currently selected context node, click **OK**. If you wish to select another context node, do so in the schema tree and then click **OK**.
4. In the case of some design elements, such as Auto-Calculations, a further step is required, such as the definition of an Auto-Calculation. In other cases, such as the insertion of a user-defined template, the Insert Design Element dialog is skipped. In such cases, another dialog, such as the Edit XPath Expression dialog will pop up. Carry out the required step and press the dialog's **OK** button.

The design element will be inserted at the end of Step 3 or Step 4, depending on the kind of design element being inserted.

Layout elements

There are three layout element commands in the Insert Design Elements toolbar: to insert (i) a layout container; (ii) a layout box; and (iii) a line. Note that layout boxes and lines can only be inserted within a layout container.

To insert a layout container, select the **Insert Layout Container** icon and then click at the location in the design where you wish to insert the layout container. You will be prompted about the size of the layout container, on selecting which the layout container will be inserted. To insert a layout box, click the **Insert Layout Box** icon, then move the cursor to the location within the layout container at which you wish to insert the layout box and click. The layout box is inserted. Click inside the layout box to start typing. To insert a line, click the **Insert Line** icon, then move the cursor to the location within the layout container at which you wish to start drawing the line. Click to define the start point of the line and then drag the cursor to the desired endpoint. Release the cursor at the end point. The line is inserted and extends from the indicated start point to the indicated end point.

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

Grid-related toggles

The **Show Grid** command toggles the display of the drawing grid on and off. When the **Snap to Grid** command is toggled on, elements created within the layout container, such as layout boxes and lines, snap to grid lines and grid line intersections. The properties of the grid can be set in the Design tab of teh Options dialog (**Tools | Options**).

 © 2012 Altova GmbH

Design Filter

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

Icon	Command	Description
	Show only one template	Shows the selected template only. Place the cursor in a template and click to show that template only.
	Show all template types	Shows all templates in the SPS (main, global, named, and layout) .
	Show imported templates	Toggles the display of imported templates on and off.
	Show/Hide main template	Toggles the display of the main template on and off.
	Show/Hide global templates	Toggles the display of global templates on and off.
	Show/Hide Design Fragments	Toggles the display of Design Fragments on and off.

The Design Filter combo box (*screenshot below*) displays a list of all the templates in the SPS.

Selecting a template in the combo box causes the template to be selected in the design. The combo box, therefore, enables you to quickly navigate to the desired template in the design, which is useful if the design has several templates, some of which might be currently hidden.

Global Resources

The **Global Resources toolbar** (*screenshot below*) enables you: (i) to select the active configuration for the application, and (ii) to access the Altova Global Resources dialog.

Select the active configuration from among the options in the dropdown list of the combo box. Click the Manage Global Resources icon to access the Altova Global Resources dialog.

© 2012 Altova GmbH

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	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	Ctrl+O	Opens an existing SPS document.
	Save Design	Ctrl+S	Saves the active SPS document.
	Save All	Ctrl+Shift+S	Saves all open SPS documents.
	Print	Ctrl+P	Prints the Authentic View of the Working XML file.
	Print Preview		Displays a print preview of the Authentic View of the Working XML File.
	Cut	Shift+Del	Cuts the selection and places it in the clipboard.
	Copy	Ctrl+C	Copies the selection to the clipboard.
	Paste	Ctrl+P	Pastes the clipboard item to the cursor location.
	Delete	Del	Deletes the selection.
	Undo	Alt+ Backspace	Undoes an editing change. An unlimited number of Undo actions can be performed at a time.
	Redo	Ctrl+Y	Redoes an undo.
	Find	Ctrl+F	Finds text in Authentic View and Output Views.
	Find Next	F3	Finds the next occurrence of the searched text.
	XSLT 1.0		Sets XSLT 1.0 as the stylesheet language.
	XSLT 2.0		Sets XSLT 2.0 as the stylesheet language.
	Spelling		Runs a spelling check on the SPS document.

14.2 Design View

The Design View is where the SPS is structured and where presentation properties are assigned. It provides you with a graphical representation of your design. The symbols that are used to denote the various components of the SPS are important for understanding the structure and layout of the SPS. These symbols are explained in the Symbols sub-section of this section. A key mechanism used to access nodes in XML documents is XPath, and a number of StyleVision features use XPath. A dialog used in common by all these features is the Edit XPath Expression dialog, in which you can build XPath expressions. The Edit XPath Expressions dialog is explained in detail in the XPath Dialog sub-section of this section.

© 2012 Altova GmbH

Symbols

An SPS design will typically contain several types of component. Each component is represented in the design by a specific symbol. These symbols are listed below and are organized into the following groups:

- Nodes in the XML document
- XML document content
- Data-entry devices
- Predefined formats
- XPath objects
- URI objects

Each of these component types can:

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

Nodes in the XML document

Element and attribute nodes in the XML document are represented in the SPS design document by tags. Each node has a start tag and end tag. Double-clicking either the start or end tag collapses that node. When a node is collapsed all its contents are hidden. Double-clicking a collapsed node expands it and displays its content.

The following types of node are represented:

- ***Document node***

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)*.

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***

 © 2012 Altova GmbH

- ***Radio buttons***

These tags can be collapsed and expanded by double-clicking an expanded and the collapsed tag, respectively. For a detailed description of how each of these data-entry devices is used, see Data-Entry Devices.

Predefined formats
Predefined formats are shown in mauve tags, which can be expanded/collapsed by double-clicking.

The screenshot above shows tags for the predefined format `p ( para)` , expanded *(at left)* and collapsed *(at right)*. To apply a predefined format, highlight the items around which the predefined format is to appear (by clicking a component and/or marking text), and insert the predefined format.

XPath objects
StyleVision features two mechanisms that use XPath expressions:

- ***Conditional templates***

Condition tags are blue. The start tag contains cells. The leftmost cell contains a question mark. Other cells each contain either (i) a number, starting with one, for each `when` condition; and/or (ii) an asterisk for the optional `otherwise` condition. A condition branch can be selected by clicking it. The number of the selected condition branch is highlighted in the start tag, and the template for that branch is displayed (within the start and end tags of the condition). The XPath expression for the selected condition branch is also highlighted in the Design Tree. Note that tags for conditions cannot be expanded/collapsed.

- ***Auto-Calculations***

Auto-Calculations are represented in Design View by the `=( AutoCalc)` object (*see screenshot above*). The XPath expression for the selected Auto-Calculation is highlighted in the Design Tree. The dialog to edit the Auto-Calculation is accessed via the Properties sidebar.

URI objects
There are three URI-based objects that can be inserted in a design:

- ***Images***
  If an image is inserted in the SPS design and can be accessed by StyleVision, it becomes visible in Design View. If it cannot be accessed, its place in the SPS is marked by an image placeholder.

- ***Bookmarks (Anchors)***

  Bookmark tags are yellow and indicated with the character A (*screenshots above*). A bookmark is created with the command **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 link item can be created 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*).

 © 2012 Altova GmbH

Edit XPath Expression

The **Edit XPath Expression** dialog (*screenshot below*) is used to edit and assign XPath expressions for a range of features.

In the Edit XPath Expression dialog, you can (i) enter an expression in the Expression text box via the keyboard, or (ii) you can insert nodes, operators, and functions by double-clicking them from their respective lists. XPath axes are listed under operators, and XML Schema constructor functions under functions. The lists for operators and functions automatically displays XPath 1.0 operators and functions or XPath 2.0 operators and functions according to the XSLT version selected for the SPS (XPath 1.0 for XSLT 1.0, and XPath 2.0 for XSLT 2.0). If you enter a part of the expression incorrectly, this will be displayed with a red underline, and in the case of spelling errors, correct alternatives will appear in a popup.

The Edit XPath Expression dialog helps you to build XPath expressions in the following ways.

- **Context node**
  The context node for the XPath expression is shown in the Selection text box in the Select Schema Attribute or Element pane. The Condition, Auto-Calculation, etc, for which the expression is being created, will be inserted at a location within this context, and the XPath expression will be evaluated with this node as its context.

- **Inserting a node from the schema in the expression**
  In the Select Schema Attribute or Element pane, the entire schema is displayed. You can insert a node from the schema into the XPath expression by double-clicking the required node. If the Absolute XPath check box is not checked, the selected node will be inserted with a location path expression that is relative to the context node. For

example, in the screenshot above, the `Location` element, which is a child of the `Office` element (the context node), has been inserted with a location path that is relative to the context node (that is, as `Location`). If the Absolute XPath check box were checked, the `Location` node would have been inserted as `/OrgChart/Office/Location`.

- **Inserting XPath operators**
  The Select Operator pane automatically lists XPath 1.0 or XPath 2.0 operators according to whether XSLT 1.0 or XSLT 2.0 has been selected as the XSLT version for the SPS. To insert an operator in the XPath expression, double-click the required operator.

- **Namespace information**
  The schema tree in the Select Schema Node 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 functions**
  The Select Function pane (*screenshot below*) is at the right of the Edit XPath Expression dialog and automatically lists XPath 1.0 or XPath 2.0 functions according to whether XSLT 1.0 or XSLT 2.0 has been selected as the XSLT version for the SPS. 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 (see `adjust-date-to-timezone` in screenshot below). Arguments in a signature are separated by commas, and arguments can have an occurrence indicators (`?` 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 functions list also includes the return type of that function and a brief description of the function.

  To insert a function in the XPath expression, double-click the required function.

- Java and .NET extension functions can be used in XPath expressions, enabling you to access the functions of these programming languages. The **Java** and **.NET** buttons at the bottom of the dialog, pop up info boxes with explanations about how to use Java and .NET extension functions in XPath expressions. For more information about this, see the Extension Functions section of this **documentation**.

Note: **Java and .NET extension functions** are not supported in the Community Edition of Altova's Authentic View products. They are supported in the Enterprise Editions of these products.

 © 2012 Altova GmbH

XPath expression entry options
As an expression is being entered into the Expression text box, the available options are displayed in a popup (*screenshot below*).

These include elements (such as `presswatch` in the screenshot above), descendant nodes (`presswatch/selection` in the screenshot above), XPath functions (`fn: upper-case` above) and XPath axes (`ancestor-or-self` above). The list of available options becomes more restricted as the expression is entered in the Expression text box.

The Otherwise check box
The Otherwise check box below the input field for the XPath expression appears when a second or subsequent condition is being added to a conditional template. Checking the Otherwise check box inserts the optional Otherwise condition of a conditional template. For details of how to use the Otherwise condition, see Conditional Templates.

XPath expressions containing carriage returns / linefeeds
You can include carriage returns and/or linefeeds (CR/LFs) in the XPath expression in order to set part of the output on separate lines. However, in order for the CR/LF to be visible in the output, the component containing the XPath expression must be enclosed in the `pre` special paragraph type. An example of such an XPath expression is:

```
translate('a;b;c', ';', codepoints-to-string(13))
```

14.3 File Menu

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

- New, to create a new SPS from a variety of sources.
- Open, Reload, Close, Close All, to open and close the active file, and to reload the active file.
- Save Design, Design As, All, which are commands to save the active SPS and all open SPS files.
- Save Generated Files, to save output files that can be generated using the SPS.
- Web Design, generates all the files required to run an ASPX application, in the folder location you specify.
- Properties, to set the encoding of the output documents, the CSS compatibility mode of the browser, how relative image paths in Authentic View should be resolved, and whether images should be embedded or linked in the RTF (*Enterprise and Professional editions*) and Word 2007+ (*Enterprise edition only*) outputs.
- Print Preview, Print, enabled in output views, these commands print what is displayed in the previews.
- Most Recently Used Files, Exit, respectively, to select a recently used file to open, and to exit the program.

 © 2012 Altova GmbH

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 two types:

- A new SPS file based on an XML Schema or DTD or XML Schema generated from an XML file (**New from XML Schema / DTD / XML**). The selected schema is added to the Design Overview sidebar and a graphical tree representation is added to the schema tree (in the Schema Tree sidebar). In Design View, the SPS is created with an empty main template. A new SPS can also be created from a file (schema or XML) via a URL or global resource (*see below*).
- A new SPS based on a user-defined schema you create node-by-node from an HTML file (**New from HTML File**). The user-defined schema is added to the Design Overview sidebar and Schema Tree sidebar. In the schema tree, it will have a single document element (root element), and the HTML file is loaded in Design View.
- An SPS can be created from an XSLT-for-HTML or an XSLT-for-FO or an FO file. Template structure and styling in the XSLT will be created in the SPS. You can then modify the SPS components and add content and formatting to the SPS. See New from XSLT for details.
- A new empty SPS (**New (empty)**). No schema is added to either the Design Overview sidebar or the schema tree. An empty main template will be created in Design View.

Note: A global resource can be used to locate a file or DB resource.

Selecting the type of design

After you have selected (XSD and XML) sources files, if required, the Create New Design dialog appears.

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

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

In a form-based document, a single Layout Container is created, in which design components can be positioned absolutely. The dimensions of the Layout Container are user-defined, and Layout Boxes can be positioned absolutely within the Layout Container and document content can be placed within individual Layout Boxes. If you wish the design of your SPS to replicate a specific form-based design, you can use an image of the original form as a blueprint image. The blueprint image can then be included as the background image of the Layout Container. The blueprint image is used to help you design your form; it will not be included in the output.

Selecting files via URLs and Global Resources

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

 © 2012 Altova GmbH

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.

 © 2012 Altova GmbH

The file URL appears in the File URL field (*screenshot above*). The **Open** button only becomes active at this point.

6. Click the **Open** button to load the file. The file you open appears in the main window.

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

Note: To give you more control over the loading process, you can choose to load the file through the local cache or a proxy server (which considerably speeds up the process if the file has been loaded before). Alternatively, you may want to reload the file if you are working, say, with an electronic publishing or database system; select the **Reload** option in this case

Microsoft® SharePoint® Server Notes

Note the following points about files on Microsoft® SharePoint® Servers:

- In the directory structure that appears in the Available Files pane (*screenshot below*), file icons have symbols that indicate the check-in/check-out status of files.

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

- The various file icons are shown below:

	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**.

 © 2012 Altova GmbH

Opening and saving files via Global Resources
To open or save a file via a global resources, click **Switch to Global Resource**. This pops up a dialog in which you can select the global resource. These dialogs are described in the section, Using Global Resources. For a general description of Global Resources, see the Global Resources section in this documentation.

Open, Reload, Close, Close All

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

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

The **Close** command closes the currently active SPS document. Note that while several files can be open, only one is active. The active document can also be closed by clicking the **Close** button at the top right of the Main Window. If you have unsaved changes in the document, you will be prompted to save these changes.

The **Close All** command closes all the open SPS documents. If you have unsaved changes in an open document, you will be prompted to save these changes.

Selecting 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.

 © 2012 Altova GmbH

Selecting files via URLs

To select a file via a URL, do the following:

1. Click the **Switch to URL** command. This switches to the URL mode of the Open dialog (*screenshot below*).

2. Enter the URL you want to access, in the *Server URL* field (*screenshot above*). If the server is a Microsoft® SharePoint® Server, check the *Microsoft® SharePoint® Server* check box. See the Microsoft® SharePoint® Server Notes below for further information about working with files on this type of server.
3. If the server is password protected, enter your User-ID and password in the *User* and *Password* fields.
4. Click **Browse** to view and navigate the directory structure of the server.
5. In the folder tree, browse for the file you want to load and click it.

The file URL appears in the File URL field (*screenshot above*). The **Open** button only becomes active at this point.

6. Click the **Open** button to load the file. The file you open appears in the main window.

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

Note: To give you more control over the loading process, you can choose to load the file through the local cache or a proxy server (which considerably speeds up the process if the file has been loaded before). Alternatively, you may want to reload the file if you are working, say, with an electronic publishing or database system; select the **Reload** option in this case

Microsoft® SharePoint® Server Notes

Note the following points about files on Microsoft® SharePoint® Servers:

- In the directory structure that appears in the Available Files pane (*screenshot below*), file icons have symbols that indicate the check-in/check-out status of files.

 © 2012 Altova GmbH

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

- The various file icons are shown below:

	Checked in. Available for check-out.
	Checked out by another user. Not available for check-out.
	Checked out locally. Can be edited and checked-in.

- After you check out a file, you can edit it in your Altova application and save it using **File | Save (Ctrl+S)**.
- You can check-in the edited file via the context menu in the Open URL dialog (*see screenshot above*), or via the context menu that pops up when you click the file tab in the Main Window of your application (*screenshot below*).

- When a file is checked out by another user, it is not available for check out.
- When a file is checked out locally by you, you can undo the check-out with the Undo Check-Out command in the context menu. This has the effect of returning the file unchanged to the server.
- If you check out a file in one Altova application, you cannot check it out in another Altova application. The file is considered to be already checked out to you. The available commands at this point in any Altova application supporting Microsoft® SharePoint® Server will be: **Check In** and **Undo Check Out**.

Opening and saving files via Global Resources
To open or save a file via a global resources, click **Switch to Global Resource**. This pops up a dialog in which you can select the global resource. These dialogs are described in the section, Using Global Resources. For a general description of Global Resources, see the Global Resources section in this documentation.

© 2012 Altova GmbH

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 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.

 © 2012 Altova GmbH

The file URL appears in the File URL field (*screenshot above*). The **Open** button only becomes active at this point.

6. Click the **Open** button to load the file. The file you open appears in the main window.

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

Note: To give you more control over the loading process, you can choose to load the file through the local cache or a proxy server (which considerably speeds up the process if the file has been loaded before). Alternatively, you may want to reload the file if you are working, say, with an electronic publishing or database system; select the **Reload** option in this case

Microsoft® SharePoint® Server Notes

Note the following points about files on Microsoft® SharePoint® Servers:

- In the directory structure that appears in the Available Files pane (*screenshot below*), file icons have symbols that indicate the check-in/check-out status of files.

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

- The various file icons are shown below:

	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**.

 © 2012 Altova GmbH

Opening and saving files via Global Resources
To open or save a file via a global resources, click **Switch to Global Resource**. This pops up a dialog in which you can select the global resource. These dialogs are described in the section, Using Global Resources. For a general description of Global Resources, see the Global Resources section in this documentation.

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** command. The advantage of using the **Save As** command is that files that have already been saved with a filename can be saved with another filename.

 © 2012 Altova GmbH

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.

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

The Save Generated HTML File command generates an HTML file. This operation requires two input files:

- The Working XML File assigned to the currently active SPS file. If no Working XML File has been assigned, 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.

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.

© 2012 Altova GmbH

Properties

The **Properties** command pops up the Properties dialog, 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 for all subsequently created SPS documents; this is done in the Encoding tab of the Options dialog.

- ***HTML output mode:*** You can select whether an 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 dialog).

XSLT

In the XSLT tab, the XSLT version for the active document can be selected. Checking the *Use xsl:import-schema declaration* option causes the `xsl:import-schema` element of the XSLT 2 specification to be included in the XSLT 2.0 document generated by StyleVision. It is recommended that you use 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.

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**.

 © 2012 Altova GmbH

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.

14.4 Edit Menu

The **Edit** menu contains commands that aid the editing of SPS documents. Besides the standard editing commands, such as **Cut** (Shift+Del), **Copy** (Ctrl+C), **Paste** (Ctrl+V), and **Delete** (Del), which are not described in this section, the following commands are available:

- Undo, Redo, Select All, to undo or restore your previous actions, and to select all content of the SPS.
- Find, Find Next, Replace, to find text in the SPS and XSLT stylesheet previews.
- Stylesheet Parameters, to edit parameters declared globally for the SPS.
- Collapse/Expand Markup, to collapse and expand SPS design component tags.

Commands are also available via the context menu which appears when you right-click a component or right-click at a cursor insertion point. Additionally, some commands are available as keyboard shortcuts and/or toolbar icons. Note, however, that commands which are not applicable in a particular document view or at a given location are grayed out in the menu.

© 2012 Altova GmbH

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.

Find, Find Next, Replace

The **Find (Ctrl+F)** command allows you to find words or fragments of words in the Design View, JavaScript Editor, and XSLT-for-HTML stylesheet.

Design View, HTML Preview,
Clicking the **Find** command in Design View, HTML Preview, pops up the following dialog:

Note the following:

- In Design View, the static data is searched, but not node names.
- To match the entry with whole words, check "Match whole word only". For example, an entry of `soft` will find only the whole word `soft`; it will not find, for example, the `soft` in `software`.
- To match the entry with fragments of words, leave the "Match whole word only" check box unchecked. Doing this would enable you, for example, to enter `soft` and `software`.
- To make the search case-insensitive, leave the "Match case" checkbox unchecked. This would enable you to find, say, `Soft` with an entry of `soft`.

XSLT-for-HTML and JavaScript Editor
Clicking the **Find** command in the XSLT-for-HTML or JavaScript Editor tab pops up the following dialog:

The following points should be noted:

- To enter a regular expression as the search term, check the Regular expression check box. You can create a regular expression with the help of a menu that pops out when you click the right-pointing arrowhead near the search term entry field.

 © 2012 Altova GmbH

- To set restrictions on what part of the document to search, click the Advanced button. This makes more search options available (*screenshot below*):

Select the types of document content you wish to search by checking the appropriate check box.

Find Next command

The **Find Next (F3)** command repeats the last Find command to search for the next occurrence of the requested text. See Find for a description of how to use the search function.

Replace (Ctrl+H)

The **Replace** command is enabled in Design View, JavaScript Editor, and Authentic View (*not supported in Basic edition*) and enables you to search for a text string and replace it with another text string.

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.

 © 2012 Altova GmbH

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.

14.5 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.

 © 2012 Altova GmbH

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 f toolbars, see the section Reference | Toolbars.

Status Bar

The display of the Status Bar, which is located at the bottom of the application window, can be switched on or off by clicking the **Status Bar** toggle command.

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.

 © 2012 Altova GmbH

Design Filter, Zoom

Design Filter

The **Design Filter** menu item rolls out a sub-menu containing commands that enable you to filter the templates that are displayed in Design View. This is useful if your design is very long or contains several templates. Using the Design Filter mechanism, you can specify what kinds of template to display. The following filter options are available:

Icon	Command	Description
	Show only one template	Shows the selected template only. Place the cursor in a template and click to show that template only.
	Show all template types	Shows all templates in the SPS (main, global, named, and layout) .
	Show imported templates	Toggles the display of imported templates on and off.
	Show/Hide main template	Toggles the display of the main template on and off.
	Show/Hide global templates	Toggles the display of global templates on and off.
	Show/Hide Design Fragments	Toggles the display of Design Fragments on and off.

Note that these commands are also available as toolbar icons in the Design Filters 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.

14.6 Insert Menu

The **Insert** menu provides commands enabling you to insert a variety of design components into the SPS. Some of these commands are available as toolbar icons. Additionally, **Insert** menu commands are also available via context menus which appear when, in the SPS design, you right-click a cursor insertion point. In the context menus, commands that are not available at that location in the SPS are disabled.

Note: Since the **Insert** commands are used for constructing the SPS, they are available in Design View only.

© 2012 Altova GmbH

Contents

The **Contents** command inserts a `( content)` placeholder at the cursor location point. There `( content)` placeholder can be inserted within two types of node, **element** and **attribute**, and it indicates that all children of the current node will be processed.

- If the current node is an element node, the node's children element nodes and text nodes will be processed. For the processing of children element nodes, global templates will be used if these exist. Otherwise the built-in template rule for elements will be used. For the processing of text nodes, the built-in template rule for text nodes will be used, the effect of which is to output the text. Effectively, the built-in template rule for elements, outputs the text of all descendant text nodes. It is important to note that the values of attributes will not be output when the `( content)` placeholder is used—unless a global template is defined for the attribute's parent element or one of its ancestors and the attribute is explicitly output, using either the `( content)` placeholder or any other content-rendering component.
- If the current node is an attribute node, the built-in template rule for the attribute's child text node will be used. This template copies the text of the text node to the output, effectively outputting the attribute's value.

The `( content)` placeholder can also be inserted for a node by placing the cursor inside the node tags, right-clicking, and selecting **Insert | Contents** or by clicking the **Insert Contents** icon in the Insert Design Elements toolbar, and then clicking the location in the design where the element is to be inserted.

Styling the contents

The `( content)` placeholder can be formatted by selecting it and using a predefined format and/or properties in Styles sidebar. This formatting is visible in the design, and, in the output, it will be applied to the contents of the node.

Replacing contents

If another node from the schema tree is dropped into a node containing a `( content)` placeholder, then the existing `( content)` placeholder is replaced by the new node.

Deleting contents

The `( content)` placeholder can be deleted by selecting it and pressing the **Delete** key on the keyboard.

Note: You can create an **empty template rule** by deleting the `( content)` placeholder of a node. An empty template rule is useful if you wish to define that some node have no template applied to it, i.e. produce no output.

Rest of Contents

The **Rest of Contents** command inserts the `(rest-of-contents)` placeholder for that node. This placeholder represents the content of **unused child nodes** of the current node; it corresponds to the `xsl:apply-templates` rule of XSLT applied to the unused elements and text nodes of the current element. Note that templates are not applied for child attributes. the `(rest-of-contents)` placeholder can also be inserted for an element by placing the cursor inside the element tags, right-clicking, and selecting **Insert | Rest of Contents**.

Use the `(rest-of-contents)` placeholder in situations where you wish to process one child element in a specific way and apply templates to its siblings. It is important to apply templates to siblings in order to avoid the possibility that the siblings are not processed. This enables you to reach elements lower down in the document hierarchy.

The `(rest-of-contents)` placeholder can be deleted by selecting it and pressing the **Delete** key on the keyboard.

 © 2012 Altova GmbH

Form Controls

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

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

Form controls can also be inserted in the design by right-clicking at the insertion point and selecting **Insert | Contents**, or by clicking the respective Form Control icon in the Insert Design Elements toolbar, and then clicking the location in the design where the element is to be inserted.

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

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.

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 (*screenshot below*) pops up.

The context node for the expression being built is highlighted in the schema tree in the pane at extreme left. You can enter the XPath expression directly in the text box, or you can double click an item (in any of the three panes) to insert that item. Nodes inserted from the schema tree in the left-hand pane are inserted relative to the context node (if the Absolute XPath check box is unchecked) or as an absolute expression starting from the document node (if the Absolute

 © 2012 Altova GmbH

XPath check box is checked).

After completing the XPath expression, click **OK** to finish inserting the Auto-Calculation.

Paragraph, Special Paragraph

The **Paragraph** command inserts an HTML paragraph `<p>` element around the selected component. A component is considered selected for this purpose when the entire node is selected (by clicking either of its tags) or when static text is selected. If the cursor is placed within static text, the paragraph element is inserted (start and end tags) at this point. A paragraph can also be inserted by using the **Insert Paragraph** icon in the Insert Design Elements toolbar.

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

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

 © 2012 Altova GmbH

Image

The **Image** command allows you to insert an image using an image location address that either comes from the XML document (dynamic) or is entered by you directly in the SPS (static).

To insert an image, do the following:

1. Click **Insert | Image** or the Insert Image toolbar icon. The Insert Image dialog (*shown below*) appears. An image can also be inserted by using the **Insert Image** icon in the Insert Design Elements toolbar.

2. Select the required tab (Static, Dynamic, or Static and Dynamic), and enter the address of the image location and/or the XPath expression that locates the image address in the XML document. The screenshot above shows how a Static and Dynamic address is entered.

Using unparsed entities
If the SPS is DTD-based and uses unparsed entities, then, for the dynamic part of an image address, the URI declared as the value of the unparsed entity can be used. For details of how to use unparsed entities, see Unparsed Entity URIs.

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.

 © 2012 Altova GmbH

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 and Dynamic SPS Tables.

Note that tables can also be created by using the **Table | Insert Table** menu command and the **Insert Table** icon in the Insert Design Elements toolbar.

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.

 © 2012 Altova GmbH

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 as required. A dynamic list can also be created by dragging a node from the Schema Tree into the design.

 © 2012 Altova GmbH

Bookmark

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

To insert a bookmark, do the following:

1. Place the cursor at the location where you wish to create the bookmark.
2. Select **Insert | Bookmark**, or right-click and select **Insert | Bookmark**. The Insert Bookmark dialog appears.

3. In the Insert Bookmark dialog, select a tab according to whether the name of the bookmark should be static (Static tab), dynamically obtained from the XML document (Dynamic), or composed of both static and dynamic parts (Static and Dynamic). In the screenshot above a dynamic bookmark is created, which has a name that is a unique ID for each `Name` child of the context node.
4. Click **OK**. The bookmark is defined.

Note: Bookmarks are created at the location specified in the design. If that location is within an element that repeats, a bookmark is created within each instance of that repeating element. If a static name is given, then each bookmark will have the same name. Therefore, it is better in such cases (of repeating elements) to give a dynamic name, which can be, for example, the name of a child element of the context node (the element within which the bookmark is created). If the node selected for the dynamic name might have the same content across multiple instances, then the uniqueness of the bookmark name can be ensured by using the `generate-id()` function to generate the name (*see screenshot above*). To reference such a bookmark, the same ID can be generated as the `href` value of a hyperlink. In this case make sure you use the fragment-identifier `#` in front of the `generate-id()` function. The XPath expression would be: `concat('#', generate-id(nodeXXX)).`

You can edit the name of a bookmark after it has been created. Do this by right-clicking the bookmark and selecting the **Edit Bookmark Name** command from the context menu that appears. Alternatively, in the Properties sidebar, in the *Link* group of properties for the link, 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. Alternatively, select the link in the Properties sidebar and click the **Delete** button in the toolbar of the sidebar.

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.
3. In the Insert Hyperlink dialog that appears, specify the document or document fragment you wish to link to. If you are linking to a document fragment (that is, to a bookmark within a document, remember to include the `#` symbol. The URI for the hyperlink is specified in one of the following forms:
   - As a static address (entered directly; you can select an HTML file via the **Browse** button, and a fragment in the current document via the **Bookmark** button). Examples would be: `http://www.altova.com` (static Web page URI); `U:\documentation\index.html` (via Browse button); or `#top_of_page` (via Bookmark button).
   - As a dynamic address (which comes from a node in the XML document; you specify the node). An example would be a node such as `//otherdocs/doc1`. If the name of a bookmark has been generated using the `generate-id()` function, then the `href` of the hyperlink should be generated using the same `generate-id()` function. For information, see Defining Hyperlinks.
   - As a combination of static and dynamic text for an address (you specify the static text and the XML document node). An example would be `www.altova.com -- department/name -- #intropara`.
4. Click **OK**. The hyperlink is created.

Note: When specifying the node for a dynamic hyperlink entry, you can enter the XPath expression as an absolute XPath expression by checking the Absolute Path check box. If this check box is not checked, the XPath expression for the node you select via the Schema button is entered as being relative to the currently selected component.

Using unparsed entities

For the dynamic part of a hyperlink address, you can use the URI declared for an unparsed entity in the DTD—if you are using a DTD. For details of how to use unparsed entities, see Using unparsed entity URIs.

Removing 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.

 © 2012 Altova GmbH

Deleting a hyperlink
To delete a hyperlink, select it in the design and press the **Delete** key. Alternatively, select the link in the Properties sidebar and click the **Delete** button in the toolbar of the sidebar.

Condition, Output-Based Condition

The **Condition** command enables you to insert a condition at the cursor point or around the selection. A condition consists of one or more branches, with each branch containing a specific set of processing rules. In this way, different sets of processing rules can be specified for different branches. For example, if the content of a node is the string `Stop`, the branch can test this, and specify that the contents of the node be colored red; a second branch can test whether the contents of the node is the string `Go`, and, if yes, color the contents of the node green; a third branch can specify that if the contents of the node is neither the string `Stop` nor the string `Go`, the contents of the node should be colored black.

To insert a condition, do the following:

1. Place the cursor at the desired location in the design or select the component around which the condition is to be inserted.
2. Select the menu command **Insert | Condition** or right-click and select the context menu command **Insert | Condition**.
3. In the Edit XPath Expression dialog that pops up (*screenshot below*), enter the XPath expression.

The context node for the expression being built is highlighted in the schema tree in the pane at extreme left. You can enter the XPath expression directly in the text box, or you can double click an item (in any of the three panes) to insert that item. Nodes inserted from the schema tree in the left-hand pane are inserted relative to the context node (if the Absolute XPath check box is unchecked) or as an absolute expression starting from the document node (if the Absolute XPath check box is checked).

4. Click **OK** to finish inserting the condition. The condition is created with one branch, the test for which is the XPath expression you entered.

Editing the XPath expressions of branches

 © 2012 Altova GmbH

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.

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).

 © 2012 Altova GmbH

User-Defined Template

The **User-Defined Template** command inserts, at the cursor insertion point, an empty template that selects a node the user specifies in an XPath expression. Insert a user-defined template as follows.

1. Place the cursor in the design at the location where the template is to be inserted.
2. Click the **Insert | User-Defined Template** command. This pops up the Edit XPath Expression dialog.
3. Enter the XPath expression to select the node you want. There are a few points to note in this connection: (i) The XPath expression will be evaluated in the context of the node within which the user-defined template is being created; (ii) The XPath expression can select any node anywhere in the document as well as in another XML document.
4. After you have entered the XPath expression, click **OK** to finish.

An empty user-defined template for the targeted node will be created.

For more detailed information, see the section, SPS File: Contents | User-Defined Templates.

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.

To insert a variable template, do the following:

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

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

 © 2012 Altova GmbH

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.

Layout Container, Layout Box, Line

The **Insert | Layout Container** command enables a Layout Container to be inserted anywhere in the design. A Layout Box and a Line can be inserted in a Layout Container, and both these commands are enabled only when a Layout Container is selected.

Layout Containers, Layout Boxes, and Lines can also be inserted via the respective icons in the Insert Design Elements toolbar. To insert via the toolbar icons, you must first select the appropriate toolbar icon and then click in the design at the location where you wish to insert the layout item.

For a detailed description of Layout modules and how to insert and use them in the design, see the section Layout Modules.

 © 2012 Altova GmbH

Table of Contents

Mousing over the **Table of Contents** command rolls out a submenu containing commands to insert various commands relating to the creation of a Table of Contents (TOC) template, TOC bookmarks, and a design document structure for the TOC.

The list of commands is as follows. For the details of how to use them click on the respective links, which will take you to the section on how to use that particular TOC component.

- Insert Table of Contents
- TOC Bookmark
- TOC Bookmark (Wizard)
- TOC Reference
- TOC Reference | Entry Text / Leader / Page Reference
- Hierarchical Numbering
- Sequential Numbering
- Level
- Level Reference
- Template Serves as Level

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

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.

 © 2012 Altova GmbH

User-Defined Item

Mousing over the **Insert | User-Defined Item** command causes a sub-menu to roll out that contains commands to insert a User-Defined Element or a User-Defined XML Text Block. How to use these two components is described in the section SPS File: Content | User-Defined Elements, XML Text Blocks.

14.7 Enclose With Menu

The **Enclose witht** menu provides commands enabling you to enclose a selection in the design with a variety of design components. Some of these commands are available as toolbar icons that enable you to insert the component in the design (equivalent commands are available in the **Insert menu**). Additionally, **Enclose with** menu commands are also available via context menus which appear when, in the SPS design, you right-click a selection. In the menus and context menus, commands that are not available at that location in the SPS are disabled.

Note: Since the **Enclose with** commands are used for constructing the SPS, they are available in Design View only.

© 2012 Altova GmbH

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.

User-Defined Template

The **Enclose with | User-Defined Template** command encloses the selection with a template for a node the user specifies in an XPath expression. Insert a user-defined template as follows.

1. Select the component in the design that you wish to enclose with a user-defined template.
2. Click the **Enclose with | User-Defined Template** command. This pops up the Edit XPath Expression dialog.
3. Enter the XPath expression to select the node you want. There are a few points to note in this connection: (i) The XPath expression will be evaluated in the context of the node within which the user-defined template is being created; (ii) The XPath expression can select any node anywhere in the document as well as in another XML document.
4. After you have entered the XPath expression, click **OK** to finish.

A user-defined template for the targeted node will be created around the selection.

For more information, see the section, SPS File: Structure | Templates and Design Fragments | Variable Templates.

 © 2012 Altova GmbH

Variable Template

The **Enclose with | Variable Template** command encloses the selection with a template for a variable defined in the SPS design.

1. Select the component in the design that you wish to enclose with a variable template.
2. Click the **Enclose with | Variable Template** command. This pops up the Enclose with Variable Template dialog.
3. From the list in the dialog, select the variable for which you wish to create the template.
4. Click **OK** to finish.

A variable template will be created around the selection.

For more information, see the section, SPS File: Structure | Templates and Design Fragments | Variable Templates.

Paragraph, Special Paragraph

The **Paragraph** command inserts an HTML paragraph `<p>` element around the selected component. A component is considered selected for this purpose when the entire node is selected (by clicking either of its tags) or when static text is selected. If the cursor is placed within static text, the paragraph element is inserted (start and end tags) at this point. A paragraph can also be inserted by using the **Insert Paragraph** icon in the Insert Design Elements toolbar.

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

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

 © 2012 Altova GmbH

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.

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.

© 2012 Altova GmbH

Condition, Output-Based Condition

The **Condition** command enables you to insert a condition at the cursor point or around the selection. A condition consists of one or more branches, with each branch containing a specific set of processing rules. In this way, different sets of processing rules can be specified for different branches. For example, if the content of a node is the string `Stop`, the branch can test this, and specify that the contents of the node be colored red; a second branch can test whether the contents of the node is the string `Go`, and, if yes, color the contents of the node green; a third branch can specify that if the contents of the node is neither the string `Stop` nor the string `Go`, the contents of the node should be colored black.

To insert a condition, do the following:

1. Place the cursor at the desired location in the design or select the component around which the condition is to be inserted.
2. Select the menu command **Insert | Condition** or right-click and select the context menu command **Insert | Condition**.
3. In the Edit XPath Expression dialog that pops up (*screenshot below*), enter the XPath expression.

The context node for the expression being built is highlighted in the schema tree in the pane at extreme left. You can enter the XPath expression directly in the text box, or you can double click an item (in any of the three panes) to insert that item. Nodes inserted from the schema tree in the left-hand pane are inserted relative to the context node (if the Absolute XPath check box is unchecked) or as an absolute expression starting from the document node (if the Absolute XPath check box is checked).

4. Click **OK** to finish inserting the condition. The condition is created with one branch, the test for which is the XPath expression you entered.

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.

 © 2012 Altova GmbH

TOC Bookmarks and TOC Levels

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

- TOC Bookmark
- TOC Bookmark (Wizard)
- Level
- Level Reference

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

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.

 © 2012 Altova GmbH

User-Defined Element

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

14.8 Table Menu

The **Table** menu provides commands enabling you to insert a static or dynamic table and to change the structure and properties of static and dynamic tables. You can edit table structure by appending, inserting, deleting, joining, and splitting rows and columns. Properties of the table as well as of individual columns, rows, and cells are defined using CSS styles and HTML properties for tables and its sub-components.

The Table commands are available in the **Table** menu (*see list below*) and as icons in the Table toolbar. The availability of various table commands depends on the current cursor position. A table can be inserted at any location in the SPS by clicking the Insert Table command. To edit the table structure, place the cursor in the appropriate cell, column, or row, and select the required editing command. To edit a formatting property, place the cursor in the appropriate cell, column, row, or table, and, in the Styles sidebar and/or Properties sidebar, define the required property for that table component.

The following commands are available in the Table menu:

- Insert Table, Delete Table
- Add Table Headers, Footers
- Append/Insert Row/Column
- Delete Row, Column
- Join Cell Left, Right, Below, Above
- Split Cell Horizontally, Vertically
- View Cell Bounds, Table Markup
- Table Properties
- Vertical Alignment of Cell Content

Headers and footers
When you create a dynamic table, you can specify whether you wish to include headers and/or footers. (Footers are allowed only when the table grows top–down.) You can create a header and footer in a static table by manually inserting a top and bottom row, respectively. The structures of headers and footers in both static and dynamic tables can be modified by splitting and joining cells.

Navigating in tables
Use the Tab and arrow keys to navigate the table cells.

Adding cell content
Any type of SPS component can be inserted as the content of a cell. The component should be formatted using the standard formatting tools.

 © 2012 Altova GmbH

Insert Table, Delete Table

The **Insert Table** command inserts an empty static table into the design tab. Selecting this command opens a dialog box in which you select whether you wish to create a static or dynamic table.

- If you choose to create a static table, a dialog prompts you for the size of the table (in terms of its rows and columns).
- If you choose to create a dynamic, the XPath Selector dialog pops up, in which you can select the node that is to be created as a dynamic table. On clicking **OK**, the Create Dynamic Table dialog pops up, in which you can select the child nodes you wish to display as the fields of each table item. For details, see Creating dynamic tables.

You can change the structure of a table subsequently by appending, inserting, and deleting rows and/or columns.

The **Delete Table** command deletes the static or dynamic table in which the cursor is.

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.

 © 2012 Altova GmbH

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.

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.

© 2012 Altova GmbH

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.

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.

© 2012 Altova GmbH

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 tese 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.

Table Properties

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

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

 © 2012 Altova GmbH

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.

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.

© 2012 Altova GmbH

14.9 Properties Menu

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

- Bullets and Numbering command, to insert lists.
- Predefined Format Strings command, to define numeric datatype formats for a given SPS.

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.

© 2012 Altova GmbH

Predefined Value Formatting Strings

Any `( content)` placeholder, input field, or Auto-Calculation which is of a `numeric`, `date`, `time`, `dateTime` or `duration` datatype can be assigned a custom format with the Value Formatting dialog. In the Value Formatting dialog, you can either create a format directly or select from a drop-down list of predefined formats.

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

- Predefined formats that have been delivered with StyleVision, and
- Predefined formats that the user creates with the **Predefined Value Formatting Strings** command (this command). When a user creates predefined value formats, these are created for the currently open SPS file—not for the entire application. After the user creates predefined value formats, the SPS file must be saved in order for the formats to be available when the file is next opened.

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

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

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. If you need help with the syntax, use the **Insert Field** and **Field Options** buttons.

3. After you have defined a format, click **OK** and save the SPS file. The formatting string is added to the list of predefined formats for that datatype, and it will appear as an option in the Value Formatting dialog (of the current SPS file) when the selected element is of the corresponding datatype.

Note:

- You can add as many custom format strings for different datatypes as you want.
- The sequential order of format strings in the Predefined Format Strings dialog determines the order in which these format strings appear in the Value Formatting dialog. The customized format strings appear above the supplied predefined formats.
- To edit a custom format string, double-click the entry in the Predefined Format Strings dialog.
- To delete a custom format string, select it, and click the **Delete** icon in the Predefined Value Formatting Strings dialog.

 © 2012 Altova GmbH

14.10 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
- Spelling Options
- Global Resources
- Active Configuration
- Customize

Spelling

The **Spelling (Shift+F7)** 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.

After all the words not found in the dictionary have been displayed in turn, and an action taken for each, the spell-checker displays the message: "The spelling check is complete." You can then recheck the document from the beginning (**Recheck Document**) or close the dialog (**Close**).

The **Options** button opens the Spelling Options dialog, in which you can specify options for the spelling check.

 © 2012 Altova GmbH

Spelling Options

The **Spelling options** command opens a dialog box (shown below) in which you specify options for the spelling check.

Always suggest corrections:
Selecting 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:
Selecting 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:
Selecting this option causes all upper case words to be ignored.

Ignore words with numbers:
Selecting this option causes all words containing numbers to be ignored.

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.
Dictionaries

Adding dictionaries for the spellchecker
For each dictionary language there are two Hunspell dictionary files that work together: a `.aff` file and `.dic` file. All language dictionaries are installed in a `Lexicons` folder at the following location:

On Windows 7: `C:\ProgramData\Altova\SpellChecker\Lexicons`
On Windows XP: `C:\Documents and Settings\All Users\Application`

`Data\Altova\SharedBetweenVersions\SpellChecker\Lexicons`

Within the `Lexicons` folder, different language dictionaries are each stored in different folder: `<language name>\<dictionary files>`. For example, on a Windows 7 machine, files for the two English-language dictionaries (`English (British)` and `English (US)`) will be stored as below:

```
C:\ProgramData\Altova\SpellChecker\Lexicons\English (British)\en_GB.aff
C:\ProgramData\Altova\SpellChecker\Lexicons\English (British)\en_GB.dic
C:\ProgramData\Altova\SpellChecker\Lexicons\English (US)\en_US.dic
C:\ProgramData\Altova\SpellChecker\Lexicons\English (US)\en_US.dic
```

In the Spelling Options dialog, the dropdown list of the *Dictionary Language* combo box displays the language dictionaries. These dictionaries are those available in the `Lexicons` folder and have the same names as the language subfolders in the `Lexicons` folder. For example, in the case of the English-language dictionaries shown above, the dictionaries would appear in the Dictionary Language combo box as: *English (British)* and *English (US)*.

All installed dictionaries are shared by the different users of the machine and the different major versions of Altova products (whether 32-bit or 64-bit).

You can add dictionaries for the spellchecker in two ways, neither of which require that the files be registered with the system:

- By adding Hunspell dictionaries into a new subfolder of the `Lexicons` folder. Hunspell dictionaries can be downloaded, for example, from http://wiki.services.openoffice.org/wiki/Dictionaries or http://extensions.services.openoffice.org/en/dictionaries. (Note that OpenOffice uses the zipped `OXT` format. So change the extension to `.zip` and unzip the `.aff` and `.dic` file to the language folders in the `Lexicons` folder. Also note that Hunspell dictionaries are based on Myspell dictionaries. So Myspell dictionaries can also be used.)
- By using the Altova dictionary installer, which installs a package of multiple language dictionaries by default to the correct location on your machine. The installer can be downloaded by clicking the link in the Dictionary language pane of the Spelling Options dialog (*see screenshot below*).

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 screenshot at top of section*).

 © 2012 Altova GmbH

To add a word to the user dictionary, enter the word in the Word text box and click **Add**. The word will be added to the alphabetical list in the Dictionary pane. To delete a word from the dictionary, select the word in the Dictionary pane and click **Delete**. The word will be deleted from the Dictionary pane. When you have finished editing the User Dictionary dialog, click **OK** for the changes to be saved to the user dictionary.

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

The user dictionary is located at:

On Windows 7: `C:\Users\<user>\Documents\Altova\SpellChecker\Lexicons\user.dic`
On Windows XP: `C:\Documents and Settings\<user>\My Documents\Altova\SpellChecker\Lexicons\user.dic`

Global Resources

The **Global Resources** command pops up the Altova Global Resources dialog (*screenshot below*), in which you can:

- Specify the Altova Global Resources XML File to use for global resources.
- Add file, folder, and database global resources (or aliases)
- Specify various configurations for each global resource (alias). Each configuration maps to a specific resource.

How to define global resources is described in detail in the section, Defining Global Resources.

Note: The Altova Global Resources dialog can also be accessed via the Global Resources toolbar (**View | Toolbars | Global Resources**).

 © 2012 Altova GmbH

Active Configuration

Mousing over the **Active Configuration** menu item rolls out a submenu containing all the configurations defined in the currently active Global Resources XML File (*screenshot below*).

The currently active configuration is indicated with a bullet. In the screenshot above the currently active configuration is `Default`. To change the active configuration, select the configuration you wish to make active.

Note: The active configuration can also be selected via the Global Resources toolbar (**View | Toolbars | Global Resources**).

Customize

The customize command lets you customize StyleVision to suit your personal needs.

Commands

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

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.

 © 2012 Altova GmbH

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.

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
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 text 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
The **Menu** tab allows you to customize the main menu bar as well as the context menus (right-click menus).

 © 2012 Altova GmbH

To customize a menu

1. Select the menu bar you want to customize (Default Menu currently).
2. Click the **Commands** tab, and drag the commands to the menu bar of your choice.

To delete commands from a menu

1. Click right on the command or icon representing the command, and
2. Select the **Delete** option from the popup menu,

   or,

1. Select **Tools | Customize** to open the Customize dialog box, and
2. Drag the command away from the menu and drop it as soon as the check mark icon appears below the mouse pointer.

To reset either of the menu bars

1. Select the Default Menu entry in the combo box)
2. Click the **Reset** button just below the menu name. A prompt appears asking if you are sure you want to reset the menu bar.

To customize a context menu (a right-click menu)

1. Select the context menu from the combo box.
2. Click the **Commands** tab and drag the commands to the context menu that is now open.

To delete commands from a context menu

1. Click right on the command or icon representing the command, and
2. Select the **Delete** option from the popup menu

   or

1. Select **Tools | Customize** to open the Customize dialog box, and
2. Drag the command away from the context menu and drop it as soon as the check mark icon appears below the mouse pointer.

To reset a context menu

1. Select the context menu from the combo box, and
2. Click the **Reset** button just below the context menu name. A prompt appears asking if

you are sure you want to reset the context menu.

To close a context menu window

- Click on the **Close icon** at the top right of the title bar, or
- Click the Close button of the Customize dialog box.

Menu animations

The menu animation option specifies the way a menu is displayed when a menu is clicked. Select an option from the drop-down list of menu animations.

Menu shadows

If you wish to have menus displayed with a shadow around it, select this option. All menus will then have a shadow.

Options

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.

 © 2012 Altova GmbH

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.

Options

The **Options** command opens a dialog (*screenshot below*) in which you can specify the encoding of the HTML output file.

Design View options

In the Design tab (*screenshot below*), you can set the application-wide general options for designs.

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

 © 2012 Altova GmbH

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 know. This option typically applies to User-Defined Templates in which the template has been created for items that cannot be placed in context in the schema source of the design. If a node is created within such a user-defined template, then the node can be created with (i) only its name, or (ii) with the full path to it from the schema root. You can set one of these options as the default behavior, or, alternatively, ask to be prompted each time this situation arises. The default selection for this option is *Always Ask*.

Schema Tree options
In the Schema Tree, elements and attributes can be listed alphabetically in ascending order. To do this, check the respective check boxes in the Schema 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 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.

14.11 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.

 © 2012 Altova GmbH

14.12 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, which lets you enter your license key-code once you have purchased the product.

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

- Table of Contents, Index, Search
- Activation, Order Form, Registration, Updates
- Other Commands

Table of Contents, Index, Search

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

The **Index** command opens the onscreen help manual for 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, you are presented with a list of the topics to choose from.

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

 © 2012 Altova GmbH

Activation, Order Form, Registration, Updates

Software Activation
After you download your Altova product software, you can activate it using either a free evaluation key or a purchased permanent license key.

- ***Free evaluation key.*** When you first start the software after downloading and installing it, the Software Activation dialog will pop up. In it is a button to request a free evaluation key-code. Enter your name, company, and e-mail address in the dialog that appears, and click Request Now! The evaluation key is sent to the e-mail address you entered and should reach you in a few minutes. Now enter the key in the key-code field of the Software Activation dialog box and click **OK** to start working with your Altova product. The software will be unlocked for a period of 30 days.
- ***Permanent license key.*** The Software Activation dialog contains a button to purchase a permanent license key. Clicking this button takes you to Altova's online shop, where you can purchase a permanent license key for your product. There are two types of permanent license: single-user and multi-user. Both will be sent to you by e-mail. A *single-user license* contains your license-data and includes your name, company, e-mail, and key-code.A *multi-user license* contains your license-data and includes your company name and key-code. Note that your license agreement does not allow you to install more than the licensed number of copies of your Altova software on the computers in your organization (per-seat license). Please make sure that you enter the data required in the registration dialog exactly as given in your license e-mail.

Note: When you enter your license information in the Software Activation dialog, ensure that you enter the data exactly as given in your license e-mail. For multi-user licenses, each user should enter his or her own name in the Name field.

The Software Activation dialog can be accessed at any time by clicking the **Help | Software Activation** command.

Order Form
When you are ready to order a licensed version of the software product, you can use either the **Order license key** button in the Software Activation dialog (*see previous section*) or the **Help | Order Form** command to proceed to the secure Altova Online Shop.

Registration
The first time you start your Altova software after having activated it, a dialog appears asking whether you would like to register your product. There are three buttons in this dialog:

- **OK:** Takes you to the Registration Form
- **Remind Me Later:** Pops up a dialog in which you can select when you wish to be next reminded.
- **Cancel:** Closes the dialog and suppresses it in future. If you wish to register at a later time, you can use the **Help | Registration** command.

Check for Updates
Checks with the Altova server whether a newer version than yours is currently available and displays a message accordingly.

Other Commands

The **Support Center** command is 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.

The **FAQ on the Web** command is 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.

The **Components Download** command is a link to Altova's Component Download Center on the Internet. From here you will be able to 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.

The **StyleVision on the Internet** command is a link to the Altova website on the Internet. You can learn more about StyleVision and related technologies and products at the Altova website.

The **StyleVision Training** command is a link to the Online Training page at the Altova website. Here you can select from online courses conducted by Altova's expert trainers.

The **About StyleVision** command displays the splash window and version number of your product.

 © 2012 Altova GmbH

Chapter 15

Appendices

15 Appendices

These appendices contain (i) information about the XSLT Engines used in StyleVision; (ii) information about the conversion of DB datatypes to XML Schema datatypes; (iii) technical information about StyleVision; and (iv) licensing information for StyleVision. Each appendix contains the sub-sections listed below:

XSLT Engine Information
Provides implementation-specific information about the Altova XSLT Engines, which are used by StyleVision to generate output.

- Altova XSLT 1.0 Engine
- Altova XSLT 2.0 Engine
- XPath 2.0 and XQuery 1.0 Functions
- Extension Functions (Java, .NET, and MSXSL)

Technical Data
Provides technical information about StyleVision.

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

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

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

 © 2012 Altova GmbH

15.1 XSLT Engine Information

This section contains information about implementation-specific features of the Altova XSLT 1.0 Engine and Altova XSLT 2.0 Engine.

XSLT 1.0 Engine: Implementation Information

The Altova XSLT 1.0 Engine is built into Altova's XMLSpy, StyleVision, Authentic, and MapForce XML products. It is also available in the free AltovaXML package. The Altova XSLT 1.0 Engine implements and conforms to the World Wide Web Consortium's XSLT 1.0 Recommendation of 16 November 1999 and XPath 1.0 Recommendation of 16 November 1999 . Limitations and implementation-specific behavior are listed below.

Limitations

- The **`xsl:preserve-space`** and **`xsl:strip-space`** elements are not supported.
- 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 directly as special characters; they are not inserted as HTML character references in the output. For instance, the character ` ` (the decimal character reference for a non-breaking space) is not inserted as ` ` in the HTML code, but directly as a non-breaking space.

Implementation's handling of whitespace-only nodes in source XML document
The XML data (and, consequently, the XML Infoset) that is passed to the Altova XSLT 1.0 Engine is stripped of boundary-whitespace-only text nodes. (A boundary-whitespace-only text node is a whitespace-only text node that occurs between two elements within an element of mixed content.) This stripping may have an effect on the value returned by the `fn:position()`, `fn:last()`, and `fn:count()` functions.

For any node selection that selects text nodes also, boundary-whitespace-only text nodes would typically also be included in the selection. However, since the XML Infoset used by the Altova engines has boundary-whitespace-only text nodes stripped from it, these nodes are not present in the XML Infoset. As a result, the size of the selection and the numbering of nodes in the selection will be different than that for a selection which included these text nodes. The `fn:position()`, `fn:last()`, and `fn:count()` functions, therefore, could produce results that are different from those produced by some other processors.

A situation in which boundary-whitespace-only text nodes are evaluated as siblings of other elements arises most commonly when `xsl:apply-templates` is used to apply templates. When the `fn:position()`, `fn:last()`, and `fn:count()` functions are used in patterns with a name test (for example, `para[3]`, which is short for `para[position()=3]`), boundary-whitespace-only nodes are irrelevant since only the named elements (`para` in the above example) are selected. (Note, however, that boundary-whitespace-only nodes **are** relevant in patterns that use the wildcard, for example, `*[10]`.)

Note: If a boundary-whitespace-only text node is required in the output, then insert the required whitespace within one of the two adjoining child elements. For example, the XML fragment:

```
<para>This is <b>bold</b> <i>italic</>.</para>
```

when processed with the XSLT template

```
<xsl:template match="para">
   <xsl:apply-templates/>
</xsl:template>
```

will produce:

```
This is bolditalic.
```

To get a space between `bold` and `italic` in the output, insert a space character within either

 © 2012 Altova GmbH

the `<b>` or `<i>` elements in the XML source. For example:

```
<para>This is <b>bold</b> <i> italic</i>. </para> or
<para>This is <b>bold&#x20; </b> <i>italic</i>. </para> or
<para>This is <b>bold</b><i>&#x20; italic</i>. </para>
```

When any of the `para` elements above is processed with the same XSLT template given above, it will produce:

```
This is bold italic.
```

XSLT 2.0 Engine: Implementation Information

The Altova XSLT 2.0 Engine is built into Altova's XMLSpy, StyleVision, Authentic, and MapForce XML products. It is also available in the free AltovaXML package. This section describes the engine's implementation-specific aspects of behavior. It starts with a section giving general information about the engine, and then goes on to list the implementation-specific behavior of XSLT 2.0 functions.

For information about implementation-specific behavior of XPath 2.0 functions, see the section, XPath 2.0 and XQuery 1.0 Functions.

 © 2012 Altova GmbH

General Information

The Altova XSLT 2.0 Engine conforms to the World Wide Web Consortium's (W3C's) XSLT 2.0 Recommendation of 23 January 2007. Note the following general information about the engine.

Backwards Compatibility

The Altova XSLT 2.0 Engine is backwards compatible. The only time the backwards compatibility of the XSLT 2.0 Engine comes into play is when using the XSLT 2.0 Engine of Altova XML 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.

In all other Altova products, the backwards-compatibility issue never arises. This is because these products automatically select the appropriate engine for the transformation. For example, consider that in XMLSpy you specify that a certain XML document be processed with an XSLT 1.0 stylesheet. When the transformation command is invoked, XMLSpy automatically selects the XSLT 1.0 Engine of XMLSpy to carry out the transformation.

Note: The stylesheet version is specified in the `version` attribute of the `stylesheet` or `transform` element of the stylesheet.

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 Altova 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`).
- With the CRs of 23 January 2007, the `untypedAtomic` and duration datatypes (`dayTimeDuration` and `yearMonthDuration`), which were formerly in the XPath Datatypes namespace (typically prefixed `xdt:`) have been moved to the XML Schema namespace.

- 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 Altova XSLT 2.0 Engine is schema-aware.

Whitespace in XML document
By default, the Altova XSLT 2.0 Engine strips all boundary whitespace from boundary-whitespace-only nodes in the source XML document. The removal of this whitespace affects the values that the `fn:position()`, `fn:last()`, `fn:count()`, and `fn:deep-equal()` functions return. For more details, see Whitespace-only Nodes in XML Document in the XPath 2.0 and XQuery 1.0 Functions section.

Note: If a boundary-whitespace-only text node is required in the output, then insert the required whitespace within one of the two adjoining child elements. For example, the XML fragment:

```
<para>This is <b>bold</b> <i>italic</>.</para>
```

when processed with the XSLT template

```
<xsl:template match="para">
   <xsl:apply-templates/>
</xsl:template>
```

will produce:

```
This is bolditalic.
```

To get a space between `bold` and `italic` in the output, insert a space character within either the `<b>` or `<i>` elements in the XML source. For example:

```
<para>This is <b>bold</b> <i> italic</>.</para>  or
<para>This is <b>bold&#x20;</b> <i>italic</>.</para>  or
<para>This is <b>bold</b><i>&#x20;italic</>.</para>
```

When such an XML fragment is processed with the same XSLT template given above, it will produce:

```
This is bold italic.
```

XSLT 2.0 elements and functions
Limitations and implementation-specific behavior of XSLT 2.0 elements and functions are listed in the section XSLT 2.0 Elements and Functions.

XPath 2.0 functions
Implementation-specific behavior of XPath 2.0 functions is listed in the section XPath 2.0 and XQuery 1.0 Functions.

 © 2012 Altova GmbH

XSLT 2.0 Elements and Functions

Limitations

The **`xsl:preserve-space`** and **`xsl:strip-space`** elements are not supported.

Implementation-specific behavior

Given below is a description of how the Altova XSLT 2.0 Engine handles implementation-specific aspects of the behavior of certain XSLT 2.0 functions.

`xsl:result-document`

Additionally supported encodings are: `x-base16tobinary` and `x-base64tobinary`.

`function-available`

The function tests for the availability of in-scope functions (XSLT 2.0, XPath 2.0, and extension functions).

`unparsed-text`

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

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

XPath 2.0 and XQuery 1.0 Functions

XPath 2.0 and XQuery 1.0 functions are evaluated by:

- the **Altova XPath 2.0 Engine**, which (i) is a component of the Altova XSLT 2.0 Engine, and (ii) is used in the XPath Evaluator of Altova's XMLSpy product to evaluate XPath expressions with respect to the XML document that is active in the XMLSpy interface.
- the **Altova XQuery 1.0 Engine**.

This section describes how XPath 2.0 and XQuery 1.0 functions are handled by the Altova XPath 2.0 Engine and Altova XQuery 1.0 Engine. Only those functions are listed, for which the behavior is implementation-specific, or where the behavior of an individual function is different in any of the three environments in which these functions are used (that is, in XSLT 2.0, in XQuery 1.0, and in the XPath Evaluator of XMLSpy). Note that this section does not describe how to use these functions. For more information about the usage of functions, see the World Wide Web Consortium's (W3C's) XQuery 1.0 and XPath 2.0 Functions and Operators Recommendation of 23 January 2007.

 © 2012 Altova GmbH

General Information

Standards conformance

- The Altova XPath 2.0 Engine implements the World Wide Web Consortium's (W3C's) XPath 2.0 Recommendation of 23 January 2007. The Altova XQuery 1.0 Engine implements the World Wide Web Consortium's (W3C's) XQuery 1.0 Recommendation of 23 January 2007. The XPath 2.0 and XQuery 1.0 functions support in these two engines is compliant with the XQuery 1.0 and XPath 2.0 Functions and Operators Recommendation of 23 January 2007.
- The Altova XPath 2.0 Engine conforms to the rules of XML 1.0 (Fourth Edition) and XML Namespaces (1.0).

Default functions namespace

The default functions namespace has been set to comply with that specified in the standard. Functions can therefore be called without a prefix.

Boundary-whitespace-only nodes in source XML document

The XML data (and, consequently, the XML Infoset) that is passed to the Altova XPath 2.0 Engine and Altova XQuery 1.0 Engine is stripped of boundary-whitespace-only text nodes. (A boundary-whitespace-only text node is a child whitespace-only text node that occurs between two elements within an element of mixed content.) This stripping has an effect on the value returned by the `fn:position()`, `fn:last()`, `fn:count()`, and `fn:deep-equal()` functions.

For any node selection that selects text nodes also, boundary-whitespace-only text nodes would typically also be included in the selection. However, since the XML Infoset used by the Altova engines has boundary-whitespace-only text nodes stripped from it, these nodes are not present in the XML Infoset. As a result, the size of the selection and the numbering of nodes in the selection will be different than that for a selection which included these text nodes. The `fn:position()`, `fn:last()`, `fn:count()`, and `fn:deep-equal()` functions, therefore, could produce results that are different from those produced by some other processors.

A situation in which boundary-whitespace-only text nodes are evaluated as siblings of other elements arises most commonly when `xsl:apply-templates` is used to apply templates. When the `fn:position()`, `fn:last()`, and `fn:count()` functions are used in patterns with a name test (for example, `para[3]`, which is short for `para[position()=3]`), boundary-whitespace-only nodes are irrelevant since only the named elements (`para` in the above example) are selected. (Note, however, that boundary-whitespace-only nodes **are** relevant in patterns that use the wildcard, for example, `*[10]`.)

Numeric notation

On output, when an `xs:double` is converted to a string, scientific notation (for example, `1.0E12`) is used when the absolute value is less than 0.000001 or greater than 1,000,000. Otherwise decimal or integer notation is used.

Precision of `xs:decimal`

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

Implicit timezone

When two `date`, `time`, or `dateTime` values need to be compared, the timezone of the values being compared need to be known. When the timezone is not explicitly given in such a value, the implicit timezone is used. The implicit timezone is taken from the system clock, and its value can be checked with the `fn:implicit-timezone()` function.

Collations
Only the Unicode codepoint collation is supported. No other collations can be used. String comparisons, including for the `fn:max` and `fn:min` functions, are based on this collation.

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 `fn:in-scope-prefixes()`, `fn:namespace-uri()` and `fn:namespace-uri-for-prefix()` functions.

Static typing extensions
The optional static type checking feature is not supported.

 © 2012 Altova GmbH

Functions Support

The table below lists (in alphabetical order) the implementation-specific behavior of certain functions. The following general points should be noted:

- In general, when 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`.

Function Name	Notes
`base-uri`	• If external entities are used in the source XML document and if a node in the external entity is specified as the input node argument of the `base-uri()` function, it is still the base URI of the including XML document that is used—not the base URI of the external entity. • The base URI of a node in the XML document can be modified using the `xml:base` attribute.

`collection`	• The argument is a relative URI that is resolved against the current base URI.
	• If the resolved URI identifies an XML file, then this XML file is treated as a catalog which references a collection of files. This file must have the form:
	`<collection>`
	`  <doc href="uri-1" />`
	`  <doc href="uri-2" />`
	`  <doc href="uri-3" />`
	`</collection>`
	The files referenced by the `href` attributes are loaded, and their document nodes are returned as a sequence.
	• If the resolved URI does not identify an XML file with the catalog structure described above, then the argument string (in which wildcards such as `?` and `*` are allowed) is used as a search string. XML files with names that match the search expression are loaded, and their document nodes are returned as a sequence. See examples below.
	• XSLT example: The expression `collection("c:\MyDocs\*.xml")//Title` returns a sequence of all `DocTitle` elements in the `.xml` files in the `MyDocs` folder.
	• XQuery example: The expression `{for $i in collection(c:\MyDocs\*.xml) return element doc{base-uri($i)}}` returns the base URIs of all the `.xml` files in the `MyDocs` folder, each URI being within a `doc` element.
	• The default collection is empty.

contd./

Function Name	Notes

count	• See note on whitespace in the General Information section.
current-date, current-dateTime, current-time	• The current date and time is taken from the system clock. • The timezone is taken from the implicit timezone provided by the evaluation context; the implicit timezone is taken from the system clock. • The timezone is always specified in the result.
deep-equal	• See note on whitespace in the General Information section.
doc	• An error is raised only if no XML file is available at the specified location or if the file is not well-formed. The file is validated if a schema is available. If the file is not valid, the invalid file is loaded without schema information.
id	• In a well-formed but invalid document that contains two or more elements having the same ID value, the first element in document order is returned.
in-scope-prefixes	• Only default namespaces may be undeclared in the XML document. However, even when a default namespace is undeclared on an element node, the prefix for the default namespace, which is the zero-length string, is returned for that node.
last	• See note on whitespace in the General Information section.
lower-case	• The Unicode character set is supported.
normalize-unicode	• The normalization forms NFC, NFD, NFKC, and NFKD are supported.

contd. /

Function Name	Notes
position	• See note on whitespace in the General Information section.

 © 2012 Altova GmbH

`resolve-uri`	• If the second, optional argument is omitted, the URI to be resolved (the first argument) is resolved against the base URI from the static context, which is the URI of the XSLT stylesheet or the base URI given in the prolog of the XQuery document. • The relative URI (the first argument) is appended after the last `"/"` in the path notation of the base URI notation. • If the value of the first argument is the zero-length string, the base URI from the static context is returned, and this URI includes the file name of the document from which the base URI of the static context is derived (e.g. the XSLT or XML file).
`static-base-uri`	• The base URI from the static context is the base URI of the XSLT stylesheet or the base URI specified in the prolog of the XQuery document. • When using XPath Evaluator in the XMLSpy IDE, the base URI from the static context is the URI of the active XML document.
`upper-case`	• The Unicode character set is supported.

Extensions

There are several ready-made functions in programming languages such as Java and C# that are not available as XPath 2.0 / XQuery 1.0 functions or as XSLT 2.0 functions. A good example of such functions are 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.

Altova Engines (XSLT 1.0, XSLT 2.0, and XQuery 1.0), which are used in a number of Altova products, support the use of extension functions in Java and .NET. The Altova XSLT Engines additionally support MSXSL scripts for XSLT 1.0 and 2.0 and Altova's own extension functions.

You should note that extension functions are always called from XPath expressions. This section describes how to use extension functions and MSXSL scripts in your XSLT stylesheets. These descriptions are organized into the following sections:

- Java Extension Functions
- .NET Extension Functions
- MSXSL Scripts for XSLT
- Altova Extension Functions

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.

 © 2012 Altova GmbH

Java Extension Functions

A Java extension function can be used within an XPath or XQuery expression to invoke a Java constructor or call a Java method (static or instance).

A field in a Java class is considered to be a method without any argument. A field can be static or instance. How to access fields is described in the respective sub-sections, static and instance.

This section is organized into the following sub-sections:

- Java: Constructors
- Java: Static Methods and Static Fields
- Java: Instance Methods and Instance Fields
- Datatypes: XSLT/XQuery to Java
- Datatypes: Java to XSLT/XQuery

Form of the extension function

The extension function in the XPath/XQuery expression must have the form `prefix:fname()`.

- The `prefix:` part identifies the extension function as a Java function. It does so by associating the extension function with an in-scope namespace declaration, the URI of which must begin with `java:` (*see below for examples*). The namespace declaration should identify a Java class, for example: `xmlns:myns="java:java.lang.Math"`. However, it could also simply be: `xmlns:myns="java"` (without a colon), with the identification of the Java class being left to the `fname()` part of the extension function.
- The `fname()` part identifies the Java method being called, and supplies the arguments for the method (*see below for examples*). However, if the namespace URI identified by the `prefix:` part does not identify a Java class (*see preceding point*), then the Java class should be identified in the `fname()` part, before the class and separated from the class by a period (*see the second XSLT example below*).

Note: The class being called must be on the classpath of the machine.

XSLT example

Here are two examples of how a static method can be called. In the first example, the class name (`java.lang.Math`) is included in the namespace URI and, therefore, must not be in the `fname()` part. In the second example, the `prefix:` part supplies the prefix `java:` while the `fname()` part identifies the class as well as the method.

```
<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 and User-Defined Jar Files. Note that paths to class files not in the current directory and to all JAR files must be specified.

Note: If you wish to add a namespace to an XSLT stylesheet being generated from an SPS created in StyleVision, the namespace must be added to the top-level `schema` element of the XML Schema on which the SPS is based. Note that the following namespace declaration `xmlns:java="java"` is created automatically by default in every SPS created in StyleVision.

User-Defined Class Files

If access is via a class file, then there are two possibilities:

- The class file is in a package. The XSLT or XQuery file is in the same folder as the Java package.
- The class file is not packaged. The XSLT or XQuery file is in the same folder as the class file.
- The class file is in a package. The XSLT or XQuery file is at some random location.
- The class file is not packaged. The XSLT or XQuery file is at some random location.

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()"/>
```

 © 2012 Altova GmbH

```
   </a>
</xsl:template>

</xsl:stylesheet>
```

Class file not packaged, XSLT/XQuery file in same folder as class file

The example below calls the `getVehicleType()` method of the `Car` class of the `com.altova.extfunc` package. The `Car` class file is in the following folder location: `JavaProject/com/altova/extfunc`. The XSLT file is also in the folder `JavaProject/com/altova/extfunc`.

```
<xsl:stylesheet version="2.0"
       xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
       xmlns:xs="http://www.w3.org/2001/XMLSchema"
       xmlns:fn="http://www.w3.org/2005/xpath-functions"
       xmlns:car="java:Car" >
<xsl:output exclude-result-prefixes="fn car xsl fo xs"/>

<xsl:template match="/">
   <a>
   <xsl:value-of select="car:getVehicleType()"/>
   </a>
</xsl:template>

</xsl:stylesheet>
```

Class file packaged, XSLT/XQuery file at any location

The example below calls the `getCarColor()` method of the `Car` class of the `com.altova.extfunc` package. The `com.altova.extfunc` package is in the folder `JavaProject`. The XSLT file is at any location. In this case, the location of the package must be specified within the URI as a query string. The syntax is:

```
java:classname[?path=uri-of-package]
```

where

`java:` indicates that a user-defined Java function is being called
`uri-of-package` is the URI of the Java package
`classname` is the name of the required method's class

The class is identified in a namespace URI, and the namespace is used to prefix a method call. The example below shows how to access a class file that is located in another directory than the current directory.

```
<xsl:stylesheet version="2.0"
       xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
       xmlns:xs="http://www.w3.org/2001/XMLSchema"
       xmlns:fn="http://www.w3.org/2005/xpath-functions"
       xmlns:car="
java:com.altova.extfunc.Car?path=file:///C:/JavaProject/" >

<xsl:output exclude-result-prefixes="fn car xsl xs"/>

<xsl:template match="/">
   <xsl:variable name="myCar" select="car:new('red')" />
   <a><xsl:value-of select="car:getCarColor($myCar)"/></a>
</xsl:template>
```

```
</xsl:stylesheet>
```

Class file not packaged, XSLT/XQuery file at any location
The example below calls the `getCarColor()` method of the `Car` class of the `com.altova.extfunc` package. The `com.altova.extfunc` package is in the folder `JavaProject`. The XSLT file is at any location. The location of the class file is specified within the namespace URI as a query string. The syntax is:

```
java:classname[?path=uri-of-classfile]
```

where

`java:` indicates that a user-defined Java function is being called
`uri-of-classfile` is the URI of the folder containing the class file
`classname` is the name of the required method's class

The class is identified in a namespace URI, and the namespace is used to prefix a method call. The example below shows how to access a class file that is located in another directory than the current directory.

```
<xsl:stylesheet version="2.0"
       xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
       xmlns:xs="http://www.w3.org/2001/XMLSchema"
       xmlns:fn="http://www.w3.org/2005/xpath-functions"
       xmlns:car="
java:Car?path=file:///C:/JavaProject/com/altova/extfunc/" >

<xsl:output exclude-result-prefixes="fn car xsl xs"/>

<xsl:template match="/">
   <xsl:variable name="myCar" select="car:new('red')" />
   <a><xsl:value-of select="car:getCarColor($myCar)"/></a>
</xsl:template>

</xsl:stylesheet>
```

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

User-Defined Jar Files

If access is via a JAR file, the URI of the JAR file must be specified using the following syntax:

```
xmlns:classNS="java:classname?path=jar:uri-of-jarfile!/"
```

The method is then called by using the prefix of the namespace URI that identifies the class: `classNS:method()`

In the above:

`java:` indicates that a Java function is being called
`classname` is the name of the user-defined class
`?` is the separator between the classname and the path
`path=jar:` indicates that a path to a JAR file is being given
`uri-of-jarfile` is the URI of the jar file
`!/` is the end delimiter of the path

 © 2012 Altova GmbH

`classNS:method()` is the call to the method

Alternatively, the classname can be given with the method call. Here are two examples of the syntax:

```
xmlns:ns1="java:docx.layout.pages?path=jar:file:///c:/projects/docs/docx.jar!/
"
        ns1:main()

        xmlns:ns2="java?path=jar:file:///c:/projects/docs/docx.jar!/"
        ns2:docx.layout.pages.main()
```

Here is a complete XSLT example that uses a JAR file to call a Java extension function:

```
<xsl:stylesheet version="2.0"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:fn="http://www.w3.org/2005/xpath-functions"
        xmlns:car="java?path=jar:file:///C:/test/Car1.jar!/" >
<xsl:output exclude-result-prefixes="fn car xsl xs"/>

<xsl:template match="/">
        <xsl:variable name="myCar" select="car:Car1.new('red')" />
        <a><xsl:value-of select="car:Car1.getCarColor($myCar)"/></a>
</xsl:template>

<xsl:template match="car"/>

</xsl:stylesheet>
```

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

Java: Constructors

An extension function can be used to call a Java constructor. All constructors are called with the pseudo-function `new()`.

If the result of a Java constructor call can be implicitly converted to XPath/XQuery datatypes, then the Java extension function will return a sequence that is an XPath/XQuery datatype. If the result of a Java constructor call cannot be converted to a suitable XPath/XQuery datatype, then the constructor creates a wrapped Java object with a type that is the name of the class returning that Java object. For example, if a constructor for the class `java.util.Date` is called (`java.util.Date.new()`), then an object having a type `java.util.Date` is returned. The lexical format of the returned object may not match the lexical format of an XPath datatype and the value would therefore need to be converted to the lexical format of the required XPath datatype and then to the required XPath datatype.

There are two things that can be done with a Java object created by a constructor:

- It can be assigned to a variable:
  ```
  <xsl:variable name="currentdate" select="date:new()" xmlns:date="
  java:java.util.Date" />
  ```
- It can be passed to an extension function (*see Instance Method and Instance Fields*):
  ```
  <xsl:value-of select="date:toString(date:new())" xmlns:date="
  java:java.util.Date" />
  ```

Java: Static Methods and Static Fields

A static method is called directly by its Java name and by supplying the arguments for the method. Static fields (methods that take no arguments), such as the constant-value fields `E` and `PI`, are accessed without specifying any argument.

XSLT examples
Here are some examples of how static methods and fields can be called:

```
<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>
```

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:

 © 2012 Altova GmbH

1. An object is created with a constructor for the class `java.util.Date` (with the `date:new()` constructor).
2. This Java object is passed as the argument of the `jlang.Object.getClass` method.
3. The object obtained by the `getClass` method is passed as the argument to the `jlang.Object.toString` method.

The result (the value of `@type`) will be a string having the value: `java.util.Date`.

An instance field is theoretically different from an instance method in that it is not a Java object per se that is passed as an argument to the instance field. Instead, a parameter or variable is passed as the argument. However, the parameter/variable may itself contain the value returned by a Java object. For example, the parameter `CurrentDate` takes the value returned by a constructor for the class `java.util.Date`. This value is then passed as an argument to the instance method `date:toString` in order to supply the value of `/enrollment/@date`.

Datatypes: XPath/XQuery to Java

When a Java function is called from within an XPath/XQuery expression, the datatype of the function's arguments is important in determining which of multiple Java classes having the same name is called.

In Java, the following rules are followed:

- If there is more than one Java method with the same name, but each has a different number of arguments than the other/s, then the Java method that best matches the number of arguments in the function call is selected.
- The XPath/XQuery string, number, and boolean datatypes (*see list below*) are implicitly converted to a corresponding Java datatype. If the supplied XPath/XQuery type can be converted to more than one Java type (for example, `xs:integer`), then that Java type is selected which is declared for the selected method. For example, if the Java method being called is `fx(decimal)` and the supplied XPath/XQuery datatype is `xs:integer`, then `xs:integer` will be converted to Java's `decimal` datatype.

The table below lists the implicit conversions of XPath/XQuery string, number, and boolean types to Java datatypes.

`xs:string`	`java.lang.String`
`xs:boolean`	`boolean` (primitive), `java.lang.Boolean`
`xs:integer`	int, long, short, byte, float, double, and the wrapper classes of these, such as `java.lang.Integer`
`xs:float`	`float` (primitive), `java.lang.Float`, `double` (primitive)
`xs:double`	`double` (primitive), `java.lang.Double`
`xs:decimal`	`float` (primitive), `java.lang.Float`, `double` (primitive), `java.lang.Double`

Subtypes of the XML Schema datatypes listed above (and which are used in XPath and XQuery) will also be converted to the Java type/s corresponding to that subtype's ancestor type.

In some cases, it might not be possible to select the correct Java method based on the supplied information. For example, consider the following case.

- The supplied argument is an `xs:untypedAtomic` value of 10 and it is intended for the method `mymethod(float)`.
- However, there is another method in the class which takes an argument of another datatype: `mymethod(double)`.
- Since the method names are the same and the supplied type (`xs:untypedAtomic`) could be converted correctly to either `float` or `double`, it is possible that `xs:untypedAtomic` is converted to `double` instead of `float`.
- Consequently the method selected will not be the required method and might not produce the expected result. To work around this, you can create a user-defined method with a different name and use this method.

Types that are not covered in the list above (for example `xs:date`) will not be converted and will generate an error. However, note that in some cases, it might be possible to create the required Java type by using a Java constructor.

Datatypes: Java to XPath/XQuery

When a Java method returns a value, the datatype of the value is a string, numeric or boolean type, then it is converted to the corresponding XPath/XQuery type. For example, Java's `java.lang.Boolean` and `boolean` datatypes are converted to `xsd:boolean`.

One-dimensional arrays returned by functions are expanded to a sequence. Multi-dimensional arrays will not be converted, and should therefore be wrapped.

When a wrapped Java object or a datatype other than string, numeric or boolean is returned, you can ensure conversion to the required XPath/XQuery type by first using a Java method (e.g `toString`) to convert the Java object to a string. In XPath/XQuery, the string can be modified to fit the lexical representation of the required type and then converted to the required type (for example, by using the `cast as` expression).

 © 2012 Altova GmbH

.NET Extension Functions

If you are working on the .NET platform, you can use extension functions written in any of the .NET languages (for example, C#). A .NET extension function can be used within an XPath or XQuery expression to invoke a constructor, property, or method (static or instance) within a .NET class.

A property of a .NET class is called using the syntax `get_PropertyName()`.

This section is organized into the following sub-sections:

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

Form of the extension function

The extension function in the XPath/XQuery expression must have the form `prefix:fname()`.

- The `prefix:` part is associated with a URI that identifies the .NET class being addressed.
- The `fname()` part identifies the constructor, property, or method (static or instance) within the .NET class, and supplies any argument/s, if required.
- The URI must begin with `clitype:` (which identifies the function as being a .NET extension function).
- The `prefix:fname()` form of the extension function can be used with system classes and with classes in a loaded assembly. However, if a class needs to be loaded, additional parameters containing the required information will have to be supplied.

Parameters

To load an assembly, the following parameters are used:

`asm`	The name of the assembly to be loaded.
`ver`	The version number (maximum of four integers separated by periods).
`sn`	The key token of the assembly's strong name (16 hex digits).
`from`	A URI that gives the location of the assembly (DLL) to be loaded. If the URI is relative, it is relative to the XSLT or XQuery document. If this parameter is present, any other parameter is ignored.
`partialname`	The partial name of the assembly. It is supplied to `Assembly.LoadWith.PartialName()`, which will attempt to load the assembly. If `partialname` is present, any other parameter is ignored.
`loc`	The locale, for example, `en-US`. The default is `neutral`.

If the assembly is to be loaded from a DLL, use the `from` parameter and omit the `sn` parameter. If the assembly is to be loaded from the Global Assembly Cache (GAC), use the `sn` parameter and omit the `from` parameter.

A question mark must be inserted before the first parameter, and parameters must be separated by a semi-colon. The parameter name gives its value with an equals sign (*see example below*).

Examples of namespace declarations

An example of a namespace declaration in XSLT that identifies the system class `System.Environment`:

```
xmlns:myns="clitype:System.Environment"
```

An example of a namespace declaration in XSLT that identifies the class to be loaded as `Trade.Forward.Scrip`:

```
xmlns:myns="clitype:Trade.Forward.Scrip?asm=forward;version=10.6.2.1"
```

An example of a namespace declaration in XQuery that identifies the system class `MyManagedDLL.testClass`:. Two cases are distinguished:

1. When the assembly is loaded from the GAC:

```
declare namespace cs="clitype:MyManagedDLL.testClass?asm=MyManagedDLL;
ver=1.2.3.4;loc=neutral;sn=b9f091b72dccfba8";
```

2. When the assembly is loaded from the DLL (complete and partial references below):

```
declare namespace
cs="clitype:MyManagedDLL.testClass?from=file:///C:/Altova
Projects/extFunctions/MyManagedDLL.dll;

declare namespace
cs="clitype:MyManagedDLL.testClass?from=MyManagedDLL.dll;
```

XSLT example

Here is a complete XSLT example that calls functions in system class `System.Math`:

```
<xsl:stylesheet version="2.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:fn="http://www.w3.org/2005/xpath-functions">
   <xsl:output method="xml" omit-xml-declaration="yes" />
   <xsl:template match="/">
      <math xmlns:math="clitype:System.Math">
         <sqrt><xsl:value-of select="math:Sqrt(9)"/></sqrt>
         <pi><xsl:value-of select="math:PI()"/></pi>
         <e><xsl:value-of select="math:E()"/></e>
         <pow><xsl:value-of select="math:Pow(math:PI(), math:E())"/></pow>
      </math>
   </xsl:template>
</xsl:stylesheet>
```

The namespace declaration on the element `math` associates the prefix `math:` with the URI `clitype:System.Math`. The `clitype:` beginning of the URI indicates that what follows identifies either a system class or a loaded class. The `math:` prefix in the XPath expressions associates the extension functions with the URI (and, by extension, the class) `System.Math`. The extension functions identify methods in the class `System.Math` and supply arguments where required.

XQuery example

Here is an XQuery example fragment similar to the XSLT example above:

```
<math xmlns:math="clitype:System.Math">
   {math:Sqrt(9)}
</math>
```

As with the XSLT example above, the namespace declaration identifies the .NET class, in this

 © 2012 Altova GmbH

case a system class. The XQuery expression identifies the method to be called and supplies the argument.

.NET: Constructors

An extension function can be used to call a .NET constructor. All constructors are called with the pseudo-function `new()`. If there is more than one constructor for a class, then the constructor that most closely matches the number of arguments supplied is selected. If no constructor is deemed to match the supplied argument/s, then a '`No constructor found`' error is returned.

Constructors that return XPath/XQuery datatypes
If the result of a .NET constructor call can be implicitly converted to XPath/XQuery datatypes, then the .NET extension function will return a sequence that is an XPath/XQuery datatype.

Constructors that return .NET objects
If the result of a .NET constructor call cannot be converted to a suitable XPath/XQuery datatype, then the constructor creates a wrapped .NET object with a type that is the name of the class returning that object. For example, if a constructor for the class `System.DateTime` is called (with `System.DateTime.new()`), then an object having a type `System.DateTime` is returned.

The lexical format of the returned object may not match the lexical format of a required XPath datatype. In such cases, the returned value would need to be: (i) converted to the lexical format of the required XPath datatype; and (ii) cast to the required XPath datatype.

There are three things that can be done with a .NET object created by a constructor:

- It can be used within a variable:
  ```
  <xsl:variable name="currentdate" select="date:new(2008, 4, 29)"
  xmlns:date="clitype:System.DateTime" />
  ```
- It can be passed to an extension function (*see Instance Method and Instance Fields*):
  ```
  <xsl:value-of select="date:ToString(date:new(2008, 4, 29))" xmlns:date
  ="clitype:System.DateTime" />
  ```
- It can be converted to a string, number, or boolean:
- ```
  <xsl:value-of select="xs:integer(data:get_Month(date:new(2008, 4, 29)))
  " xmlns:date="clitype:System.DateTime" />
  ```

.NET: Static Methods and Static Fields

A static method is called directly by its name and by supplying the arguments for the method. The name used in the call must exactly match a public static method in the class specified. If the method name and the number of arguments that were given in the function call matches more than one method in a class, then the types of the supplied arguments are evaluated for the best match. If a match cannot be found unambiguously, an error is reported.

Note: A field in a .NET class is considered to be a method without any argument. A property is called using the syntax `get_PropertyName()`.

Examples
An XSLT example showing a call to a method with one argument (`System.Math.Sin(arg)`):

```
<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>
```

.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

 © 2012 Altova GmbH

The difference between an instance method and an instance field is theoretical. In an instance method, a .NET object is directly passed as an argument; in an instance field, a parameter or variable is passed instead—though the parameter or variable may itself contain a .NET object. For example, in the example above, the variable `releasedate` contains a .NET object, and it is this variable that is passed as the argument of `ToString()` in the second `date` element constructor. Therefore, the `ToString()` instance in the first `date` element is an instance method while the second is considered to be an instance field. The result produced in both instances, however, is the same.

Datatypes: XPath/XQuery to .NET

When a .NET extension function is used within an XPath/XQuery expression, the datatypes of the function's arguments are important for determining which one of multiple .NET methods having the same name is called.

In .NET, the following rules are followed:

- If there is more than one method with the same name in a class, then the methods available for selection are reduced to those that have the same number of arguments as the function call.
- The XPath/XQuery string, number, and boolean datatypes (*see list below*) are implicitly converted to a corresponding .NET datatype. If the supplied XPath/XQuery type can be converted to more than one .NET type (for example, `xs:integer`), then that .NET type is selected which is declared for the selected method. For example, if the .NET method being called is `fx(double)` and the supplied XPath/XQuery datatype is `xs:integer`, then `xs:integer` will be converted to .NET's `double` datatype.

The table below lists the implicit conversions of XPath/XQuery string, number, and boolean types to .NET datatypes.

`xs:string`	`StringValue, string`
`xs:boolean`	`BooleanValue, bool`
`xs:integer`	`IntegerValue, decimal, long, integer, short, byte, double, float`
`xs:float`	`FloatValue, float, double`
`xs:double`	`DoubleValue, double`
`xs:decimal`	`DecimalValue, decimal, double, float`

Subtypes of the XML Schema datatypes listed above (and which are used in XPath and XQuery) will also be converted to the .NET type/s corresponding to that subtype's ancestor type.

In some cases, it might not be possible to select the correct .NET method based on the supplied information. For example, consider the following case.

- The supplied argument is an `xs:untypedAtomic` value of 10 and it is intended for the method `mymethod(float)`.
- However, there is another method in the class which takes an argument of another datatype: `mymethod(double)`.
- Since the method names are the same and the supplied type (`xs:untypedAtomic`) could be converted correctly to either `float` or `double`, it is possible that `xs:untypedAtomic` is converted to `double` instead of `float`.
- Consequently the method selected will not be the required method and might not produce the expected result. To work around this, you can create a user-defined method with a different name and use this method.

Types that are not covered in the list above (for example `xs:date`) will not be converted and will generate an error.

Datatypes: .NET to XPath/XQuery

When a .NET method returns a value and the datatype of the value is a string, numeric or boolean type, then it is converted to the corresponding XPath/XQuery type. For example, .NET's `decimal` datatype is converted to `xsd:decimal`.

When a .NET object or a datatype other than string, numeric or boolean is returned, you can ensure conversion to the required XPath/XQuery type by first using a .NET method (for example `System.DateTime.ToString()`) to convert the .NET object to a string. In XPath/XQuery, the string can be modified to fit the lexical representation of the required type and then converted to the required type (for example, by using the `cast as` expression).

 © 2012 Altova GmbH

MSXSL Scripts for XSLT

The `<msxsl:script>` element contains user-defined functions and variables that can be called from within XPath expressions in the XSLT stylesheet. The `<msxsl:script>` is a top-level element, that is, it must be a child element of `<xsl:stylesheet>` or `<xsl:transform>`.

The `<msxsl:script>` element must be in the namespace `urn:schemas-microsoft-com:xslt` (*see example below*).

Scripting language and namespace

The scripting language used within the block is specified in the `<msxsl:script>` element's `language` attribute and the namespace to be used for function calls from XPath expressions is identified with the `implements-prefix` attribute (*see below*).

```
<msxsl:script language="scripting-language" implements-prefix="user-namespace-
prefix">

   function-1 or variable-1
   ...
   function-n or variable-n

</msxsl:script>
```

The `<msxsl:script>` element interacts with the Windows Scripting Runtime, so only languages that are installed on your machine may be used within the `<msxsl:script>` element. **The .NET Framework 2.0 platform or higher must be installed for MSXSL scripts to be used.** Consequently, the .NET scripting languages can be used within the `<msxsl:script>` element.

The `language` attribute accepts the same values as the `language` attribute on the HTML `<script>` element. If the `language` attribute is not specified, then Microsoft JScript is assumed as the default.

The implements-prefix attribute takes a value that is a prefix of a declared in-scope namespace. This namespace typically will be a user namespace that has been reserved for a function library. All functions and variables defined within the `<msxsl:script>` element will be in the namespace identified by the prefix specified in the `implements-prefix` attribute. When a function is called from within an XPath expression, the fully qualified function name must be in the same namespace as the function definition.

Example

Here is an example of a complete XSLT stylesheet that uses a function defined within a `<msxsl:script>` element.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:fn="http://www.w3.org/2005/xpath-functions"
   xmlns:msxsl="urn:schemas-microsoft-com:xslt"
   xmlns:user="http://mycompany.com/mynamespace">

  <msxsl:script language="VBScript" implements-prefix="user">
    <![CDATA[
    ' Input: A currency value: the wholesale price
    ' Returns: The retail price: the input value plus 20% margin,
    ' rounded to the nearest cent
    dim a as integer  = 13
    Function AddMargin(WholesalePrice) as integer
```

```
      AddMargin = WholesalePrice * 1.2 + a
    End Function
  ]]>
  </msxsl:script>

  <xsl:template match="/">
    <html>
      <body>
        <p>
          <b>Total Retail Price =
            $<xsl:value-of select="user:AddMargin(50)"/>
          </b>
          <br/>
          <b>Total Wholesale Price =
            $<xsl:value-of select="50"/>
          </b>
        </p>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Datatypes

The values of parameters passed into and out of the script block are limited to XPath datatypes. This restriction does not apply to data passed among functions and variables within the script block.

Assemblies

An assembly can be imported into the script by using the **`msxsl:assembly`** element. The assembly is identified via a name or a URI. The assembly is imported when the stylesheet is compiled. Here is a simple representation of how the `msxsl:assembly` element is to be used.

```
<msxsl:script>
   <msxsl:assembly name="myAssembly.assemblyName" />
   <msxsl:assembly href="pathToAssembly" />

   ...

</msxsl:script>
```

The assembly name can be a full name, such as:

```
"system.Math, Version=3.1.4500.1 Culture=neutral
PublicKeyToken=a46b3f648229c514"
```

or a short name, such as `"myAssembly.Draw"`.

Namespaces

Namespaces can be declared with the **`msxsl:using`** element. This enables assembly classes to be written in the script without their namespaces, thus saving you some tedious typing. Here is how the `msxsl:using` element is used so as to declare namespaces.

```
<msxsl:script>
   <msxsl:using namespace="myAssemblyNS.NamespaceName" />

   ...

</msxsl:script>
```

The value of the `namespace` attribute is the name of the namespace.

 © 2012 Altova GmbH

Altova Extension Functions

Altova extension functions are in the namespace **http://www.altova.com/xslt-extensions** and are indicated in this section with the prefix **altova:** , which is assumed to be bound to the namespace given above.

The following extension functions are supported in the current version of your Altova product in the manner described below. However, note that in future versions of your product, support for one or more of these functions might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

General functions

- altova:evaluate()
- altova:distinct-nodes()
- altova:encode-for-rtf()
- altova:xbrl-labels()
- altova:xbrl-footnotes()
- altova:generate-auto-number()
- altova:reset-auto-number()
- altova:get-temp-folder()

General Functions

The following extension functions are supported in the current version of your Altova product in the manner described below. However, note that in future versions of your product, support for one or more of these functions might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

- altova:evaluate()
- altova:distinct-nodes()
- altova:encode-for-rtf()
- altova:xbrl-labels()
- altova:xbrl-footnotes()
- altova:generate-auto-number()
- altova:reset-auto-number()
- altova:get-temp-folder()

altova:evaluate()

The `altova:evaluate()` function takes an XPath expression, passed as a string, as its mandatory argument. It returns the output of the evaluated expression.

```
altova:evaluate(XPathExp as xs:string)
```

For example:

```
altova:evaluate('//Name[1]')
```

In the example above, note that the expression `//Name[1]` is passed as a string by enclosing it in single quotes. The `altova:evaluate` function returns the contents of the first `Name` element in the document.

 © 2012 Altova GmbH

The `altova:evaluate` function can take additional (optional) arguments. These arguments are, respectively, the values of variables with the names `p1`, `p2`, `p3`... `pN` that can be used in the XPath expression.

altova:evaluate(XPathExp as xs:string **[, p1value ... pNvalue])**

where

- the variable names must be of the form `pX`, `X` being an integer
- the sequence of the function's arguments, from the second argument onwards corresponds to the sequence of variables named `p1` to `pN`. So the second argument will be the value of the variable `p1`, the third argument that of the variable `p2`, and so on.
- The variable values must be of type `item*`

For 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 above listing, 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.

The following examples further illustrate usage:

```
<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`).

Variables can be used in the `altova:evaluate()` extension function as shown in the examples below:

- 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.

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

altova:distinct-nodes()

The `altova:distinct-nodes()` function 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`.

```
altova:distinct-nodes( $arg as node()* )  as node()*
```

altova:encode-for-rtf()

The `altova:encode-for-rtf()` function converts the input string into code for RTF.

```
altova:encode-for-rtf( $inputstr as xs:string?,
$preserveallwhitespace as xs:boolean,
$preservenewlines as xs:boolean) as xs:string
```

Whitespace and new lines will be preserved according to the boolean value specified for their respective parameters.

altova:xbrl-labels()

The `altova:xbrl-labels()` function takes two input arguments: a node name and the taxonomy file location containing the node. The function returns the XBRL labels associated with the input node.

```
altova:xbrl-labels( $name as xs:QName, $file as xs:string )  as node()*
```

altova:xbrl-footnotes()

 © 2012 Altova GmbH

The `altova:footnotes()` function takes a node as its input argument and returns the set of XBRL footnote nodes referenced by the input node.

```
altova:footnotes( $arg as node() )  as node()*
```

`altova:generate-auto-number(id as xs:string, start-with as xs:integer, increment as xs:integer, reset-on-change as xs:string)`
Generates a series of numbers having the specified ID. The start integer and the increment is specified.

`altova:reset-auto-number(id as xs:string)`
This function resets the auto-numbering of the auto-numbering series specified with the ID argument. The series is reset to the start integer of the series (see `altova:generate-auto-number` above).

`altova:get-temp-folder as xs:string`
Gets the temporary folder.

15.2 Technical Data

This section contains useful background information on the technical aspects of your software. It is organized into the following sections:

- OS and Memory Requirements
- Altova XML Parser
- Altova XSLT and XQuery Engines
- Unicode Support
- Internet Usage

© 2012 Altova GmbH

OS and Memory Requirements

Operating System
Altova software applications are:

- 32-bit Windows applications for Windows XP, Windows Server 2003 and 2008, Windows Vista, and Windows 7, or
- 64-bit Windows applications for Windows Vista and Windows 7

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

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

Altova XML Parser

When opening any XML document, the application uses its built-in validating parser (the Altova XML Parser) to check for well-formedness, validate the document against a schema (if specified), and build trees and Infosets. The Altova XML Parser 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 Altova XML Parser implements the Final Recommendation of the W3C's XML Schema specification. New developments recommended by the W3C's XML Schema Working Group are continuously being incorporated in the Altova Parser, so that Altova products give you a state-of-the-art development environment.

 © 2012 Altova GmbH

Altova XSLT and XQuery Engines

Altova products use the Altova XSLT 1.0 Engine, Altova XSLT 2.0 Engine, and Altova XQuery 1.0 Engines. Documentation about implementation-specific behavior for each engine is in the section Engine Information, in Appendix 1 of the product documentation, should that engine be used in the product.

These three engines are also available in the AltovaXML package, which can be downloaded from the Altova website free of charge. Documentation for using the engines is available with the AltovaXML package.

Unicode Support

Unicode is the 16-bit character-set (extendable to 32-bit) defined by the Unicode Consortium. It provides a unique number for every character,

- no matter what the platform,
- no matter what the program,
- no matter what the language.

Fundamentally, computers just deal with numbers. They store letters and other characters by assigning a number for each one. Before Unicode was invented, there were hundreds of different encoding systems for assigning these numbers. No single encoding could contain enough characters: for example, the European Union alone requires several different encodings to cover all its languages. Even for a single language like English, no single encoding was adequate for all the letters, punctuation, and technical symbols in common use.

These encoding systems used to conflict with one another. That is, two encodings used the same number for two different characters, or different numbers for the same character. Any given computer (especially servers) needs to support many different encodings; yet whenever data is passed between different encodings or platforms, that data always runs the risk of corruption.

Unicode is changing all that!
Unicode provides a unique number for every character, no matter what the platform, no matter what the program, and no matter what the language. The Unicode Standard has been adopted by such industry leaders as Apple, HP, IBM, JustSystems, Microsoft, Oracle, SAP, Sun, Base and many others.

Unicode is required by modern standards such as XML, Java, ECMAScript (JavaScript), LDAP, CORBA 3.0, WML, etc., and is the official way to implement ISO/IEC 10646. It is supported in many operating systems, all modern browsers, and many other products. The emergence of the Unicode Standard, and the availability of tools supporting it, are among the most significant recent global software technology trends.

Incorporating Unicode into client-server or multi-tiered applications and web sites offers significant cost savings over the use of legacy character sets. Unicode enables a single software product or a single web site to be targeted across multiple platforms, languages and countries without re-engineering. It allows data to be transported through many different systems without corruption.

 © 2012 Altova GmbH

Windows XP

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. Consequently you may encounter XML documents that contain "unprintable" characters, because the font you have selected does not contain the required glyphs. Therefore it can sometimes be very useful to have a font that covers the entire Unicode range - especially when editing XML documents from all over the world.

The most universal font we have encountered is a typeface called Arial Unicode MS that has been created by Agfa Monotype for Microsoft. This font contains over 50,000 glyphs and covers the entire set of characters specified by the Unicode 2.1 standard. It needs 23MB and is included with Microsoft Office 2000.

We highly recommend that you install this font on your system and use it with the application if you are often editing documents in different writing systems. This font is not installed with the "Typical" setting of the Microsoft Office setup program, but you can choose the Custom Setup option to install this font.

In the `/Examples` folder in your application folder you will also find a new XHTML file called `Unicode-UTF8.html` that contains the sentence "When the world wants to talk, it speaks Unicode" in many different languages ("Wenn die Welt miteinander spricht, spricht sie Unicode") and writing-systems (世界的に話すなら、Unicode です。) - this line has been adopted from the 10th Unicode conference in 1997 and is a beautiful illustration of the importance of Unicode for the XML standard. Opening this file will give you a quick impression on what is possible with Unicode and what writing systems are supported by the fonts available on your PC installation.

Right-to-Left Writing Systems

Please note that even under Windows NT 4.0 any text from a right-to-left writing-system (such as Hebrew or Arabic) is not rendered correctly except in those countries that actually use right-to-left writing-systems. This is due to the fact that only the Hebrew and Arabic versions of Windows NT contains support for rendering and editing right-to-left text on the operating system layer.

© 2012 Altova GmbH

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.
- If you use the URL mode of the Open dialog box to open a document directly from a URL (**File | Open | Switch to URL**), that document is retrieved through a http (port 80) connection. (*This functionality is available in XMLSpy and Authentic Desktop.*)
- If you open an XML document that refers to an XML Schema or DTD and the document is specified through a URL, it is also retrieved through a http (port 80) connection once you validate the XML document. This may also happen automatically upon opening a document if you have instructed the application to automatically validate files upon opening in the File tab of the Options dialog (**Tools | Options**). (*This functionality is available in XMLSpy and Authentic Desktop.*)
- 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 this manual and the Altova Software License Agreement.

15.3 License Information

This section contains:

- Information about the distribution of this software product
- Information about the intellectual property rights related to this software product
- The End User License Agreement governing the use of this software product

Please read this information carefully. It is binding upon you since you agreed to these terms when you installed this software product.

© 2012 Altova GmbH

Electronic Software Distribution

This product is available through electronic software distribution, a distribution method that provides the following unique benefits:

- You can evaluate the software free-of-charge before making a purchasing decision.
- Once you decide to buy the software, you can place your order online at the Altova website and immediately get a fully licensed product within minutes.
- When you place an online order, you always get the latest version of our software.
- The product package includes a comprehensive integrated onscreen help system. The latest version of the user manual is available at www.altova.com (i) in HTML format for online browsing, and (ii) in PDF format for download (and to print if you prefer to have the documentation on paper).

30-day evaluation period
After downloading this product, you can evaluate it for a period of up to 30 days free of charge. About 20 days into this evaluation period, the software will start to remind you that it has not yet been licensed. The reminder message will be displayed once each time you start the application. If you would like to continue using the program after the 30-day evaluation period, you have to purchase an Altova Software License Agreement, which is delivered in the form of a key-code that you enter into the Software Activation dialog to unlock the product. You can purchase your license at the online shop at the Altova website.

Helping Others within Your Organization to Evaluate the Software
If you wish to distribute the evaluation version within your company network, or if you plan to use it on a PC that is not connected to the Internet, you may only distribute the Setup programs, provided that they are not modified in any way. Any person that accesses the software installer that you have provided, must request their own 30-day evaluation license key code and after expiration of their evaluation period, must also purchase a license in order to be able to continue using the product.

For further details, please refer to the Altova Software License Agreement at the end of this section.

Software Activation and License Metering

As part of Altova's Software Activation, the software may use your internal network and Internet connection for the purpose of transmitting license-related data at the time of installation, registration, use, or update to an Altova-operated license server and validating the authenticity of the license-related data in order to protect Altova against unlicensed or illegal use of the software and to improve customer service. Activation is based on the exchange of license related data such as operating system, IP address, date/time, software version, and computer name, along with other information between your computer and an Altova license server.

Your Altova product has a built-in license metering module that further helps you avoid any unintentional violation of the End User License Agreement. Your product is licensed either as a single-user or multi-user installation, and the license-metering module makes sure that no more than the licensed number of users use the application concurrently.

This license-metering technology uses your local area network (LAN) to communicate between instances of the application running on different computers.

Single license
When the application starts up, as part of the license metering process, the software sends a short broadcast datagram to find any other instance of the product running on another computer in the same network segment. If it doesn't get any response, it will open a port for listening to other instances of the application.

Multi license
If more than one instance of the application is used within the same LAN, these instances will briefly communicate with each other on startup. These instances exchange key-codes in order to help you to better determine that the number of concurrent licenses purchased is not accidentally violated. This is the same kind of license metering technology that is common in the Unix world and with a number of database development tools. It allows Altova customers to purchase reasonably-priced concurrent-use multi-user licenses.

We have also designed the applications so that they send few and small network packets so as to not put a burden on your network. The TCP/IP ports (2799) used by your Altova product are officially registered with the IANA (*see* the IANA website (http://www.iana.org/) *for details*) and our license-metering module is tested and proven technology.

If you are using a firewall, you may notice communications on port 2799 between the computers that are running Altova products. You are, of course, free to block such traffic between different groups in your organization, as long as you can ensure by other means, that your license agreement is not violated.

You will also notice that, if you are online, your Altova product contains many useful functions; these are unrelated to the license-metering technology.

 © 2012 Altova GmbH

Intellectual Property Rights

The Altova Software and any copies that you are authorized by Altova to make are the intellectual property of and are owned by Altova and its suppliers. The structure, organization and code of the Software are the valuable trade secrets and confidential information of Altova and its suppliers. The Software is protected by copyright, including without limitation by United States Copyright Law, international treaty provisions and applicable laws in the country in which it is being used. Altova retains the ownership of all patents, copyrights, trade secrets, trademarks and other intellectual property rights pertaining to the Software, and that Altova's ownership rights extend to any images, photographs, animations, videos, audio, music, text and "applets" incorporated into the Software and all accompanying printed materials. Notifications of claimed copyright infringement should be sent to Altova's copyright agent as further provided on the Altova Web Site.

Altova software contains certain Third Party Software that is also protected by intellectual property laws, including without limitation applicable copyright laws as described in detail at http://www.altova.com/legal_3rdparty.html.

All other names or trademarks are the property of their respective owners.

Altova End User License Agreement

THIS IS A LEGAL DOCUMENT -- RETAIN FOR YOUR RECORDS

ALTOVA® END USER LICENSE AGREEMENT

Licensor:
Altova GmbH
Rudolfsplatz 13a/9
A-1010 Wien
Austria

Important - Read Carefully. Notice to User:

This End User License Agreement ("Software License Agreement") is a legal document between you and Altova GmbH ("Altova"). It is important that you read this document before using the Altova-provided software ("Software") and any accompanying documentation, including, without limitation printed materials, 'online' files, or electronic documentation ("Documentation"). By clicking the "I accept" and "Next" buttons below, or by installing, or otherwise using the Software, you agree to be bound by the terms of this Software License Agreement as well as the Altova Privacy Policy ("Privacy Policy") including, without limitation, the warranty disclaimers, limitation of liability, data use and termination provisions below, whether or not you decide to purchase the Software. You agree that this agreement is enforceable like any written agreement negotiated and signed by you. If you do not agree, you are not licensed to use the Software, and you must destroy any downloaded copies of the Software in your possession or control. You may print a copy of this Software License Agreement as part of the installation process at the time of acceptance. Alternatively, you may go to our Web site at http://www.altova.com/eula to download and print a copy of this Software License Agreement for your files and http://www.altova.com/privacy to review the Privacy Policy.

1. SOFTWARE LICENSE

(a) License Grant.

(i) Upon your acceptance of this Software License Agreement Altova grants you a non-exclusive, non-transferable (except as provided below), limited license, without the right to grant sublicenses, to install and use a copy of the Software on one compatible personal computer or workstation up to the Permitted Number of computers. Subject to the limitations set forth in Section 1(c), you may install and use a copy of the Software on more than one of your compatible personal computers or workstations if you have purchased a Named-User license. Subject to the limitations set forth in Sections 1(d) and 1(e), users may use the software concurrently on a network. The Permitted Number of computers and/or users and the type of license, e.g. Installed, Named-Users, and Concurrent-User, shall be determined and specified at such time as you elect to purchase the Software. Installed user licenses are intended to be fixed and not concurrent. In other words, you cannot uninstall for one user in order to reinstall that license to a different user and then uninstall and reinstall back to the original user. Users should be static. Notwithstanding the foregoing, permanent switchovers are acceptable (i.e., an employee has left the company, machine is retired). During the evaluation period, hereinafter defined, only a single user may install and use the software on one (1) personal computer or workstation. If you have licensed the Software as part of a suite of Altova software products (collectively, the "Suite") and have not installed each product individually, then the Software License Agreement governs your use of all of the software included in the Suite.

(ii) If you have licensed SchemaAgent, then the terms and conditions of this Software License Agreement apply to your use of the SchemaAgent server software ("SchemaAgent Server") included therein, as applicable, and you are licensed to use SchemaAgent Server solely in connection with your use of Altova Software and solely for the

 © 2012 Altova GmbH

purposes described in the accompanying documentation.

(iii) If you have licensed Software that enables users to generate source code, your license to install and use a copy of the Software as provided herein permits you to generate source code based on (i) Altova Library modules that are included in the Software (such generated code hereinafter referred to as the "Restricted Source Code") and (ii) schemas or mappings that you create or provide (such code as may be generated from your schema or mapping source materials hereinafter referred to as the "Unrestricted Source Code"). In addition to the rights granted herein, Altova grants you a non-exclusive, non-transferable, limited license to compile the complete generated code (comprised of the combination of the Restricted Source Code and the Unrestricted Source Code) into executable object code form, and to use, copy, distribute or license that executable. You may not distribute or redistribute, sublicense, sell, or transfer the Restricted Source Code to a third-party unless said third-party already has a license to the Restricted Source Code through their separate agreement with Altova. Notwithstanding anything to the contrary herein, you may not distribute, incorporate or combine with other software, or otherwise use the Altova Library modules or Restricted Source Code, or any Altova intellectual property embodied in or associated with the Altova Library modules or Restricted Source Code, in any manner that would subject the Restricted Source Code to the terms of a copyleft, free software or open source license that would require the Restricted Source Code or Altova Library modules source code to be disclosed in source code form. Altova reserves all other rights in and to the Software. With respect to the feature(s) of UModel that permit reverse-engineering of your own source code or other source code that you have lawfully obtained, such use by you does not constitute a violation of this Agreement. Except as otherwise expressly permitted in Section 1(i) reverse engineering of the Software is strictly prohibited as further detailed therein.

(iv) In the event Restricted Source Code is incorporated into executable object code form, you will include the following statement in (1) introductory splash screens, or if none, within one or more screens readily accessible by the end-user, and (2) in the electronic and/or hard copy documentation: "Portions of this program were developed using Altova® [name of Altova Software, e.g. MapForce® 2011] and include libraries owned by Altova GmbH, Copyright © 2007-2011 Altova GmbH (www.altova.com)."

(b) Server Use for Installation and Use of SchemaAgent. You may install one (1) copy of the Software on a computer file server within your internal network solely for the purpose of downloading and installing the Software onto other computers within your internal network up to the Permitted Number of computers in a commercial environment only. If you have licensed SchemaAgent, then you may install SchemaAgent Server on any server computer or workstation and use it in connection with your Software. No other network use is permitted, including without limitation using the Software either directly or through commands, data or instructions from or to a computer not part of your internal network, for Internet or Web-hosting services or by any user not licensed to use this copy of the Software through a valid license from Altova.

(c) Named-Use. If you have licensed the "Named-User" version of the software, you may install the Software on up to five (5) compatible personal computers or workstations of which you are the primary user thereby allowing you to switch from one computer to the other as necessary provided that only one (1) instance of the Software will be used by you as the Named-User at any given time. If you have purchased multiple Named-User licenses, each individual Named-User will receive a separate license key code.

(d) Concurrent Use in Same Physical Network or Office Location. If you have licensed a "Concurrent-User" version of the Software, you may install the Software on any compatible computers in a commercial environment only, up to ten (10) times the Permitted Number of users, provided that only the Permitted Number of users actually use the Software at the same time and further provided that the computers on which the Software is installed are on the same physical computer network. The Permitted Number of concurrent users shall be delineated at

such time as you elect to purchase the Software licenses. Each separate physical network or office location requires its own set of separate Concurrent User Licenses for those wishing to use the Concurrent User versions of the Software in more than one location or on more than one network, all subject to the above Permitted Number limitations and based on the number of users using the Software. If a computer is not on the same physical network, then a locally installed user license or a license dedicated to concurrent use in a virtual environment is required. Home User restrictions and limitations with respect to the Concurrent User licenses used on home computers are set forth in Section 1(g).

(e) Concurrent Use in Virtual Environment. If you have purchased Concurrent-User Licenses, you may install a copy of the Software on a terminal server (Microsoft Terminal Server, Citrix Metaframe, etc.), application virtualization server (Microsoft App-V, Citrix XenApp, VMWare ThinApp, etc.) or virtual machine environment within your internal network for the sole and exclusive purpose of permitting individual users within your organization to access and use the Software through a terminal server, application virtualization session, or virtual machine environment from another computer provided that the total number of users that access or use the Software concurrently at any given point in time on such network, virtual machine or terminal server does not exceed the Permitted Number; and provided that the total number of users authorized to use the Software through the terminal server, application virtualization session, or virtual machine environment does not exceed six (6) times the Permitted Number of users. Accordingly, the limitations set forth in Section 1(d) regarding the number of installations and the requirement that the usage be on the same physical network shall not apply to terminal server, application virtualization session, or virtual machine environments. In a virtual environment, you must deploy a means of preventing users from exceeding the Permitted Number of concurrent users. Altova makes no warranties or representations about the performance of Altova software in a terminal server, application virtualization session, or virtual machine environment and the foregoing are expressly excluded from the limited warranty in Section 5 hereof and technical support is not available with respect to issues arising from use in such environments.

(f) Backup and Archival Copies. You may make one (1) backup and one (1) archival copy of the Software, provided your backup and archival copies are not installed or used on any computer and further provided that all such copies shall bear the original and unmodified copyright, patent and other intellectual property markings that appear on or in the Software. You may not transfer the rights to a backup or archival copy unless you transfer all rights in the Software as provided under Section 3.

(g) Home Use (Personal and Non-Commercial). In order to further familiarize yourself with the Software and allow you to explore its features and functions, you, as the primary user of the computer on which the Software is installed for commercial purposes, may also install one copy of the Software on only one (1) home personal computer (such as your laptop or desktop) solely for your personal and non-commercial ("HPNC") use. This HPNC copy may not be used in any commercial or revenue-generating business activities, including without limitation, work-from-home, teleworking, telecommuting, or other work-related use of the Software. The HPNC copy of the Software may not be used at the same time on a home personal computer as the Software is being used on the primary computer.

(h) Key Codes, Upgrades and Updates. Prior to your purchase and as part of the registration for the thirty (30) day evaluation period, as applicable, you will receive an evaluation key code. You will receive a purchase key code when you elect to purchase the Software from either Altova GmbH or an authorized reseller. The purchase key code will enable you to activate the Software beyond the initial evaluation period. You may not re-license, reproduce or distribute any key code except with the express written permission of Altova. If the Software that you have licensed is an upgrade or an update, then the latest update or upgrade that you download and install replaces all or part of the Software previously licensed. The update or upgrade and the associated license keys does not constitute the granting of a second license to the Software in that you may not use the upgrade or updated copy in addition to the copy of the

 © 2012 Altova GmbH

Software that it is replacing and whose license has terminated.

(i) Title. Title to the Software is not transferred to you. Ownership of all copies of the Software and of copies made by you is vested in Altova, subject to the rights of use granted to you in this Software License Agreement. As between you and Altova, documents, files, stylesheets, generated program code (including the Unrestricted Source Code) and schemas that are authored or created by you via your utilization of the Software, in accordance with its Documentation and the terms of this Software License Agreement, are your property unless they are created using Evaluation Software, as defined in Section 4 of this Agreement, in which case you have only a limited license to use any output that contains generated program code (including Unrestricted Source Code) such as Java, C++, C#, VB.NET or XSLT and associated project files and build scripts, as well as generated XML, XML Schemas, documentation, UML diagrams, and database structures only for the thirty (30) day evaluation period.

(j) Reverse Engineering. Except and to the limited extent as may be otherwise specifically provided by applicable law in the European Union, you may not reverse engineer, decompile, disassemble or otherwise attempt to discover the source code, underlying ideas, underlying user interface techniques or algorithms of the Software by any means whatsoever, directly or indirectly, or disclose any of the foregoing, except to the extent you may be expressly permitted to decompile under applicable law in the European Union, if it is essential to do so in order to achieve operability of the Software with another software program, and you have first requested Altova to provide the information necessary to achieve such operability and Altova has not made such information available. Altova has the right to impose reasonable conditions and to request a reasonable fee before providing such information. Any information supplied by Altova or obtained by you, as permitted hereunder, may only be used by you for the purpose described herein and may not be disclosed to any third party or used to create any software which is substantially similar to the expression of the Software. Requests for information from users in the European Union with respect to the above should be directed to the Altova Customer Support Department.

(k) Other Restrictions. You may not loan, rent, lease, sublicense, distribute or otherwise transfer all or any portion of the Software to third parties except to the limited extent set forth in Section 3 or as otherwise expressly provided. You may not copy the Software except as expressly set forth above, and any copies that you are permitted to make pursuant to this Software License Agreement must contain the same copyright, patent and other intellectual property markings that appear on or in the Software. You may not modify, adapt or translate the Software. You may not, directly or indirectly, encumber or suffer to exist any lien or security interest on the Software; knowingly take any action that would cause the Software to be placed in the public domain; or use the Software in any computer environment not specified in this Software License Agreement.

You will comply with applicable law and Altova's instructions regarding the use of the Software. You agree to notify your employees and agents who may have access to the Software of the restrictions contained in this Software License Agreement and to ensure their compliance with these restrictions.

(l) THE SOFTWARE IS NEITHER GUARANTEED NOR WARRANTED TO BE ERROR-FREE NOR SHALL ANY LIABILITY BE ASSUMED BY ALTOVA IN THIS RESPECT. NOTWITHSTANDING ANY SUPPORT FOR ANY TECHNICAL STANDARD, THE SOFTWARE IS NOT INTENDED FOR USE IN OR IN CONNECTION WITH, WITHOUT LIMITATION, THE OPERATION OF NUCLEAR FACILITIES, AIRCRAFT NAVIGATION, COMMUNICATION SYSTEMS, AIR TRAFFIC CONTROL EQUIPMENT, MEDICAL DEVICES OR LIFE SUPPORT SYSTEMS, MEDICAL OR HEALTH CARE APPLICATIONS, OR OTHER APPLICATIONS WHERE THE FAILURE OF THE SOFTWARE OR ERRORS IN DATA PROCESSING COULD LEAD TO DEATH, PERSONAL INJURY OR SEVERE PHYSICAL OR ENVIRONMENTAL DAMAGE. YOU AGREE THAT YOU ARE SOLELY RESPONSIBLE FOR THE ACCURACY AND ADEQUACY OF THE SOFTWARE AND ANY DATA GENERATED OR

PROCESSED BY THE SOFTWARE FOR YOUR INTENDED USE AND YOU WILL DEFEND, INDEMNIFY AND HOLD ALTOVA, ITS OFFICERS AND EMPLOYEES HARMLESS FROM ANY 3RD PARTY CLAIMS, DEMANDS, OR SUITS THAT ARE BASED UPON THE ACCURACY AND ADEQUACY OF THE SOFTWARE IN YOUR USE OR ANY DATA GENERATED BY THE SOFTWARE IN YOUR USE.

2. INTELLECTUAL PROPERTY RIGHTS

Acknowledgement of Altova's Rights. You acknowledge that the Software and any copies that you are authorized by Altova to make are the intellectual property of and are owned by Altova and its suppliers. The structure, organization and code of the Software are the valuable trade secrets and confidential information of Altova and its suppliers. The Software is protected by copyright, including without limitation by United States Copyright Law, international treaty provisions and applicable laws in the country in which it is being used. You acknowledge that Altova retains the ownership of all patents, copyrights, trade secrets, trademarks and other intellectual property rights pertaining to the Software, and that Altova's ownership rights extend to any images, photographs, animations, videos, audio, music, text and "applets" incorporated into the Software and all accompanying printed materials. You will take no actions which adversely affect Altova's intellectual property rights in the Software. Trademarks shall be used in accordance with accepted trademark practice, including identification of trademark owners' names. Trademarks may only be used to identify printed output produced by the Software, and such use of any trademark does not give you any right of ownership in that trademark. XMLSpy®, Authentic®, StyleVision®, MapForce®, UModel®, DatabaseSpy®, DiffDog®, SchemaAgent®, SemanticWorks®, MissionKit®, Markup Your Mind®, Nanonull™, and Altova® are trademarks of Altova GmbH. (registered in numerous countries). Unicode and the Unicode Logo are trademarks of Unicode, Inc. Windows, Windows XP, Windows Vista, and Windows 7 are trademarks of Microsoft. W3C, CSS, DOM, MathML, RDF, XHTML, XML and XSL are trademarks (registered in numerous countries) of the World Wide Web Consortium (W3C); marks of the W3C are registered and held by its host institutions, MIT, INRIA and Keio. Except as expressly stated above, this Software License Agreement does not grant you any intellectual property rights in the Software. Notifications of claimed copyright infringement should be sent to Altova's copyright agent as further provided on the Altova Web Site.

3. LIMITED TRANSFER RIGHTS

Notwithstanding the foregoing, you may transfer all your rights to use the Software to another person or legal entity provided that: (a) you also transfer each of this Software License Agreement, the Software and all other software or hardware bundled or pre-installed with the Software, including all copies, updates and prior versions, and all copies of font software converted into other formats, to such person or entity; (b) you retain no copies, including backups and copies stored on a computer; (c) the receiving party secures a personalized key code from Altova; and (d) the receiving party accepts the terms and conditions of this Software License Agreement and any other terms and conditions upon which you legally purchased a license to the Software. Notwithstanding the foregoing, you may not transfer education, pre-release, or not-for-resale copies of the Software.

4. PRE-RELEASE AND EVALUATION PRODUCT ADDITIONAL TERMS

If the product you have received with this license is pre-commercial release or beta Software ("Pre-release Software"), then this Section applies. In addition, this section applies to all evaluation and/or demonstration copies of Altova software ("Evaluation Software") and continues in effect until you purchase a license. To the extent that any provision in this section is in conflict with any other term or condition in this Software License Agreement, this section shall supersede such other term(s) and condition(s) with respect to the Pre-release and/or Evaluation Software, but only to the extent necessary to resolve the conflict. You acknowledge that the Pre-release Software is a pre-release version, does not represent final product from Altova, and may contain bugs, errors and other problems that could cause system or other failures and data

loss. CONSEQUENTLY, THE PRE-RELEASE AND/OR EVALUATION SOFTWARE IS PROVIDED TO YOU **"AS-IS" WITH NO WARRANTIES FOR USE OR PERFORMANCE,** AND ALTOVA DISCLAIMS ANY WARRANTY OR LIABILITY OBLIGATIONS TO YOU OF ANY KIND, WHETHER EXPRESS OR IMPLIED. WHERE LEGALLY LIABILITY CANNOT BE EXCLUDED FOR PRE-RELEASE AND/OR EVALUATION SOFTWARE, BUT IT MAY BE LIMITED, ALTOVA'S LIABILITY AND THAT OF ITS SUPPLIERS SHALL BE LIMITED TO THE SUM OF FIFTY DOLLARS (USD $50) IN TOTAL. If the Evaluation Software has a time-out feature, then the software will cease operation after the conclusion of the designated evaluation period. Upon such expiration date, your license will expire unless otherwise extended. Your license to use any output created with the Evaluation Software that contains generated program code (including Unrestricted Source Code) such as Java, C++, C, VB.NET or XSLT and associated project files and build scripts as well as generated XML, XML Schemas, documentation, UML diagrams, and database structures terminates automatically upon the expiration of the designated evaluation period but the license to use such output is revived upon your purchase of a license for the Software that you evaluated and used to create such output. Access to any files created with the Evaluation Software is entirely at your risk. You acknowledge that Altova has not promised or guaranteed to you that Pre-release Software will be announced or made available to anyone in the future, that Altova has no express or implied obligation to you to announce or introduce the Pre-release Software, and that Altova may not introduce a product similar to or compatible with the Pre-release Software. Accordingly, you acknowledge that any research or development that you perform regarding the Pre-release Software or any product associated with the Pre-release Software is done entirely at your own risk. During the term of this Software License Agreement, if requested by Altova, you will provide feedback to Altova regarding testing and use of the Pre-release Software, including error or bug reports. If you have been provided the Pre-release Software pursuant to a separate written agreement, your use of the Software is governed by such agreement. You may not sublicense, lease, loan, rent, distribute or otherwise transfer the Pre-release Software. Upon receipt of a later unreleased version of the Pre-release Software or release by Altova of a publicly released commercial version of the Software, whether as a stand-alone product or as part of a larger product, you agree to return or destroy all earlier Pre-release Software received from Altova and to abide by the terms of the license agreement for any such later versions of the Pre-release Software.

5. LIMITED WARRANTY AND LIMITATION OF LIABILITY

(a) Limited Warranty and Customer Remedies. Altova warrants to the person or entity that first purchases a license for use of the Software pursuant to the terms of this Software License Agreement that (i) the Software will perform substantially in accordance with any accompanying Documentation for a period of ninety (90) days from the date of receipt, and (ii) any support services provided by Altova shall be substantially as described in Section 6 of this agreement. Some states and jurisdictions do not allow limitations on duration of an implied warranty, so the above limitation may not apply to you. To the extent allowed by applicable law, implied warranties on the Software, if any, are limited to ninety (90) days. Altova's and its suppliers' entire liability and your exclusive remedy shall be, at Altova's option, either (i) return of the price paid, if any, or (ii) repair or replacement of the Software that does not meet Altova's Limited Warranty and which is returned to Altova with a copy of your receipt. This Limited Warranty is void if failure of the Software has resulted from accident, abuse, misapplication, abnormal use, Trojan horse, virus, or any other malicious external code. Any replacement Software will be warranted for the remainder of the original warranty period or thirty (30) days, whichever is longer. This limited warranty does not apply to Evaluation and/or Pre-release Software.

(b) No Other Warranties and Disclaimer. THE FOREGOING LIMITED WARRANTY AND REMEDIES STATE THE SOLE AND EXCLUSIVE REMEDIES FOR ALTOVA OR ITS SUPPLIER'S BREACH OF WARRANTY. ALTOVA AND ITS SUPPLIERS DO NOT AND CANNOT WARRANT THE PERFORMANCE OR RESULTS YOU MAY OBTAIN BY USING THE SOFTWARE. EXCEPT FOR THE FOREGOING LIMITED WARRANTY, AND FOR ANY

WARRANTY, CONDITION, REPRESENTATION OR TERM TO THE EXTENT WHICH THE SAME CANNOT OR MAY NOT BE EXCLUDED OR LIMITED BY LAW APPLICABLE TO YOU IN YOUR JURISDICTION, ALTOVA AND ITS SUPPLIERS MAKE NO WARRANTIES, CONDITIONS, REPRESENTATIONS OR TERMS, EXPRESS OR IMPLIED, WHETHER BY STATUTE, COMMON LAW, CUSTOM, USAGE OR OTHERWISE AS TO ANY OTHER MATTERS. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, ALTOVA AND ITS SUPPLIERS DISCLAIM ALL OTHER WARRANTIES AND CONDITIONS, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, SATISFACTORY QUALITY, INFORMATIONAL CONTENT OR ACCURACY, QUIET ENJOYMENT, TITLE AND NON-INFRINGEMENT, WITH REGARD TO THE SOFTWARE, AND THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES. THIS LIMITED WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS. YOU MAY HAVE OTHERS, WHICH VARY FROM STATE/JURISDICTION TO STATE/JURISDICTION.

(c) Limitation of Liability. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW EVEN IF A REMEDY FAILS ITS ESSENTIAL PURPOSE, IN NO EVENT SHALL ALTOVA OR ITS SUPPLIERS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR ANY OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE THE SOFTWARE OR THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES, EVEN IF ALTOVA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN ANY CASE, ALTOVA'S ENTIRE LIABILITY UNDER ANY PROVISION OF THIS SOFTWARE LICENSE AGREEMENT SHALL BE LIMITED TO THE AMOUNT ACTUALLY PAID BY YOU FOR THE SOFTWARE PRODUCT. Because some states and jurisdictions do not allow the exclusion or limitation of liability, the above limitation may not apply to you. In such states and jurisdictions, Altova's liability shall be limited to the greatest extent permitted by law and the limitations or exclusions of warranties and liability contained herein do not prejudice applicable statutory consumer rights of person acquiring goods otherwise than in the course of business. The disclaimer and limited liability above are fundamental to this Software License Agreement between Altova and you.

(d) Infringement Claims. Altova will indemnify and hold you harmless and will defend or settle any claim, suit or proceeding brought against you by a third party that is based upon a claim that the content contained in the Software infringes a copyright or violates an intellectual or proprietary right protected by United States or European Union law ("Claim"), but only to the extent the Claim arises directly out of the use of the Software and subject to the limitations set forth in Section 5 of this Agreement except as otherwise expressly provided. You must notify Altova in writing of any Claim within ten (10) business days after you first receive notice of the Claim, and you shall provide to Altova at no cost such assistance and cooperation as Altova may reasonably request from time to time in connection with the defense of the Claim. Altova shall have sole control over any Claim (including, without limitation, the selection of counsel and the right to settle on your behalf on any terms Altova deems desirable in the sole exercise of its discretion). You may, at your sole cost, retain separate counsel and participate in the defense or settlement negotiations. Altova shall pay actual damages, costs, and attorney fees awarded against you (or payable by you pursuant to a settlement agreement) in connection with a Claim to the extent such direct damages and costs are not reimbursed to you by insurance or a third party, to an aggregate maximum equal to the purchase price of the Software. If the Software or its use becomes the subject of a Claim or its use is enjoined, or if in the opinion of Altova's legal counsel the Software is likely to become the subject of a Claim, Altova shall attempt to resolve the Claim by using commercially reasonable efforts to modify the Software or obtain a license to continue using the Software. If in the opinion of Altova's legal counsel the Claim, the injunction or potential Claim cannot be resolved through reasonable modification or licensing, Altova, at its own election, may terminate this Software License Agreement without penalty, and will refund to you on a pro rata basis any fees paid in advance by you to Altova. THE FOREGOING CONSTITUTES ALTOVA'S SOLE AND EXCLUSIVE LIABILITY FOR INTELLECTUAL

PROPERTY INFRINGEMENT. This indemnity does not apply to infringements that would not be such, except for customer-supplied elements.

6. SUPPORT AND MAINTENANCE

Altova offers multiple optional "Support & Maintenance Package(s)" ("SMP") for the version of Software product edition that you have licensed, which you may elect to purchase in addition to your Software license. The Support Period, hereinafter defined, covered by such SMP shall be delineated at such time as you elect to purchase a SMP. Your rights with respect to support and maintenance as well as your upgrade eligibility depend on your decision to purchase SMP and the level of SMP that you have purchased:

(a) If you have not purchased SMP, you will receive the Software AS IS and will not receive any maintenance releases or updates. However, Altova, at its option and in its sole discretion on a case by case basis, may decide to offer maintenance releases to you as a courtesy, but these maintenance releases will not include any new features in excess of the feature set at the time of your purchase of the Software. In addition, Altova will provide free technical support to you for thirty (30) days after the date of your purchase (the "Support Period" for the purposes of this paragraph 6(a), and Altova, in its sole discretion on a case by case basis, may also provide free courtesy technical support during your thirty (30) day evaluation period. Technical support is provided via a Web-based support form only, and there is no guaranteed response time.

(b) If you have purchased SMP, then solely for the duration of its delineated Support Period, **you are eligible to receive the version of the Software edition** that you have licensed and all maintenance releases and updates for that edition that are released during your Support Period. For the duration of your SMP's Support Period, you will also be eligible to receive upgrades to the comparable edition of the next version of the Software that succeeds the Software edition that you have licensed for applicable upgrades released during your Support Period. The specific upgrade edition that you are eligible to receive based on your Support Period is further detailed in the SMP that you have purchased. Software that is introduced as separate product is not included in SMP. Maintenance releases, updates and upgrades may or may not include additional features. In addition, Altova will provide Priority Technical Support to you for the duration of the Support Period. Priority Technical Support is provided via a Web-based support form only and Altova will make commercially reasonable efforts to respond via e-mail to all requests within forty-eight (48) hours during Altova's business hours (MO-FR, 8am UTC – 10pm UTC, Austrian and US holidays excluded) and to make reasonable efforts to provide work-arounds to errors reported in the Software.

During the Support Period you may also report any Software problem or error to Altova. If Altova determines that a reported reproducible material error in the Software exists and significantly impairs the usability and utility of the Software, Altova agrees to use reasonable commercial efforts to correct or provide a usable work-around solution in an upcoming maintenance release or update, which is made available at certain times at Altova's sole discretion.

If Altova, in its discretion, requests written verification of an error or malfunction discovered by you or requests supporting example files that exhibit the Software problem, you shall promptly provide such verification or files, by email, telecopy, or overnight mail, setting forth in reasonable detail the respects in which the Software fails to perform. You shall use reasonable efforts to cooperate in diagnosis or study of errors. Altova may include error corrections in maintenance releases, updates, or new major releases of the Software. Altova is not obligated to fix errors that are immaterial. Immaterial errors are those that do not significantly impact use of the Software as determined by Altova in its sole discretion. Whether or not you have purchased the Support & Maintenance Package, technical support only covers issues or questions resulting directly out of the operation of the Software and Altova will not provide you with generic consultation, assistance, or advice under any circumstances.

Updating Software may require the updating of software not covered by this Software License

Agreement before installation. Updates of the operating system and application software not specifically covered by this Software License Agreement are your responsibility and will not be provided by Altova under this Software License Agreement. Altova's obligations under this Section 6 are contingent upon your proper use of the Software and your compliance with the terms and conditions of this Software License Agreement at all times. Altova shall be under no obligation to provide the above technical support if, in Altova's opinion, the Software has failed due to the following conditions: (i) damage caused by the relocation of the software to another location or CPU; (ii) alterations, modifications or attempts to change the Software without Altova 's written approval; (iii) causes external to the Software, such as natural disasters, the failure or fluctuation of electrical power, or computer equipment failure; (iv) your failure to maintain the Software at Altova's specified release level; or (v) use of the Software with other software without Altova's prior written approval. It will be your sole responsibility to: (i) comply with all Altova-specified operating and troubleshooting procedures and then notify Altova immediately of Software malfunction and provide Altova with complete information thereof; (ii) provide for the security of your confidential information; (iii) establish and maintain backup systems and procedures necessary to reconstruct lost or altered files, data or programs.

7. SOFTWARE ACTIVATION, UPDATES AND LICENSE METERING

(a) License Metering. Altova has a built-in license metering module that helps you to avoid any unintentional violation of this Software License Agreement. Altova may use your internal network for license metering between installed versions of the Software.

(b) Software Activation. **Altova's 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 between your computer and the Altova license server. You agree that Altova may use these measures and you agree to follow any applicable requirements. You further agree that use of license key codes that are not or were not generated by Altova and lawfully obtained from Altova, or an authorized reseller as part of an effort to activate or use the Software violates Altova's intellectual property rights as well as the terms of this Software License Agreement. You agree that efforts to circumvent or disable Altova's copyright protection mechanisms or license management mechanism violate Altova's intellectual property rights as well as the terms of this Software License Agreement. Altova expressly reserves the rights to seek all available legal and equitable remedies to prevent such actions and to recover lost profits, damages and costs.**

(c) LiveUpdate. Altova provides a new LiveUpdate notification service to you, which is free of charge. Altova may use your internal network and Internet connection for the purpose of transmitting license-related data to an Altova-operated LiveUpdate server to validate your license at appropriate intervals and determine if there is any update available for you.

(d) Use of Data. The terms and conditions of the Privacy Policy are set out in full at http://www.altova.com/privacy and are incorporated by reference into this Software License Agreement. By your acceptance of the terms of this Software License Agreement and/or use of the Software, you authorize the collection, use and disclosure of information collected by Altova for the purposes provided for in this Software License Agreement and/or the Privacy Policy. Altova has the right in its sole discretion to amend this provision of the Software License Agreement and/or Privacy Policy at any time. You are encouraged to review the terms of the Privacy Policy as posted on the Altova Web site from time to time.

(e) Notice to European Users. Please note that the information as described in paragraph 7(d) above may be transferred outside of the European Economic Area, for purposes of processing, analysis, and review, by Altova, Inc., a company located in Beverly,

 © 2012 Altova GmbH

Massachusetts, U.S.A., or its subsidiaries or Altova's subsidiaries or divisions, or authorized partners, located worldwide. You are advised that the United States uses a sectoral model of privacy protection that relies on a mix of legislation, governmental regulation, and self-regulation. You are further advised that the Council of the European Union has found that this model does not provide "adequate" privacy protections as contemplated by Article 25 of the European Union's Data Directive. (Directive 95/46/EC, 1995 O.J. (L 281) 31). Article 26 of the European Union's Data Directive allows for transfer of personal data from the European Union to a third country if the individual has unambiguously given his consent to the transfer of personal information, regardless of the third country's level of protection. By agreeing to this Software License Agreement, you consent to the transfer of all such information to the United States and the processing of that information as described in this Software License Agreement and the Privacy Policy.

8. TERM AND TERMINATION

This Software License Agreement may be terminated (a) by your giving Altova written notice of termination; (b) by Altova, at its option, giving you written notice of termination if you commit a breach of this Software License Agreement and fail to cure such breach within ten (10) days after notice from Altova; or (c) at the request of an authorized Altova reseller in the event that you fail to make your license payment or other monies due and payable. In addition the Software License Agreement governing your use of a previous version that you have upgraded or updated of the Software is terminated upon your acceptance of the terms and conditions of the Software License Agreement accompanying such upgrade or update. Upon any termination of the Software License Agreement, you must cease all use of the Software that this Software License Agreement governs, destroy all copies then in your possession or control and take such other actions as Altova may reasonably request to ensure that no copies of the Software remain in your possession or control. The terms and conditions set forth in Sections 1(h), 1(i), 1(j), 1(k), 2, 5(b), 5(c), 5(d), 7(d), 7(e), 9, 10 and 11 survive termination as applicable.

9. RESTRICTED RIGHTS NOTICE AND EXPORT RESTRICTIONS

The Software was developed entirely at private expense and is commercial computer software provided with **RESTRICTED RIGHTS**. Use, duplication or disclosure by the U.S. Government or a U.S. Government contractor or subcontractor is subject to the restrictions set forth in this Agreement and as provided in FAR 12.211 and 12.212 (48 C.F.R. §12.211 and 12.212) or DFARS 227. 7202 (48 C.F.R. §227-7202) as applicable. Consistent with the above as applicable, Commercial Computer Software and Commercial Computer Documentation licensed to U.S. government end users only as commercial items and only with those rights as are granted to all other end users under the terms and conditions set forth in this Software License Agreement. Manufacturer is Altova GmbH, Rudolfsplatz, 13a/9, A-1010 Vienna, Austria/EU. You may not use or otherwise export or re-export the Software or Documentation except as authorized by United States law and the laws of the jurisdiction in which the Software was obtained. In particular, but without limitation, the Software or Documentation may not be exported or re-exported (i) into (or to a national or resident of) any U.S. embargoed country or (ii) to anyone on the U.S. Treasury Department's list of Specially Designated Nationals or the U.S. Department of Commerce's Table of Denial Orders. By using the Software, you represent and warrant that you are not located in, under control of, or a national or resident of any such country or on any such list.

10. THIRD PARTY SOFTWARE

The Software may contain third party software which requires notices and/or additional terms and conditions. Such required third party software notices and/or additional terms and conditions are located at our Website at http://www.altova.com/legal_3rdparty.html and are made a part of and incorporated by reference into this Agreement. By accepting this Agreement, you are also accepting the additional terms and conditions, if any, set forth therein.

11. GENERAL PROVISIONS

If you are located in the European Union and are using the Software in the European Union and not in the United States, then this Software License Agreement will be governed by and construed in accordance with the laws of the Republic of Austria (excluding its conflict of laws principles and the U.N. Convention on Contracts for the International Sale of Goods) and you expressly agree that exclusive jurisdiction for any claim or dispute with Altova or relating in any way to your use of the Software resides in the Handelsgericht, Wien (Commercial Court, Vienna) and you further agree and expressly consent to the exercise of personal jurisdiction in the Handelsgericht, Wien (Commercial Court, Vienna) in connection with any such dispute or claim.

If you are located in the United States or are using the Software in the United States then this Software License Agreement will be governed by and construed in accordance with the laws of the Commonwealth of Massachusetts, USA (excluding its conflict of laws principles and the U.N. Convention on Contracts for the International Sale of Goods) and you expressly agree that exclusive jurisdiction for any claim or dispute with Altova or relating in any way to your use of the Software resides in the federal or state courts of the Commonwealth of Massachusetts and you further agree and expressly consent to the exercise of personal jurisdiction in the federal or state courts of the Commonwealth of Massachusetts in connection with any such dispute or claim.

If you are located outside of the European Union or the United States and are not using the Software in the United States, then this Software License Agreement will be governed by and construed in accordance with the laws of the Republic of Austria (excluding its conflict of laws principles and the U.N. Convention on Contracts for the International Sale of Goods) and you expressly agree that exclusive jurisdiction for any claim or dispute with Altova or relating in any way to your use of the Software resides in the Handelsgericht, Wien (Commercial Court, Vienna) and you further agree and expressly consent to the exercise of personal jurisdiction in the Handelsgericht Wien (Commercial Court, Vienna) in connection with any such dispute or claim. This Software License Agreement will not be governed by the conflict of law rules of any jurisdiction or the United Nations Convention on Contracts for the International Sale of Goods, the application of which is expressly excluded.

This Software License Agreement contains the entire agreement and understanding of the parties with respect to the subject matter hereof, and supersedes all prior written and oral understandings of the parties with respect to the subject matter hereof. Any notice or other communication given under this Software License Agreement shall be in writing and shall have been properly given by either of us to the other if sent by certified or registered mail, return receipt requested, or by overnight courier to the address shown on Altova's Web site for Altova and the address shown in Altova's records for you, or such other address as the parties may designate by notice given in the manner set forth above. This Software License Agreement will bind and inure to the benefit of the parties and our respective heirs, personal and legal representatives, affiliates, successors and permitted assigns. The failure of either of us at any time to require performance of any provision hereof shall in no manner affect such party's right at a later time to enforce the same or any other term of this Software License Agreement. This Software License Agreement may be amended only by a document in writing signed by both of us. In the event of a breach or threatened breach of this Software License Agreement by either party, the other shall have all applicable equitable as well as legal remedies. Each party is duly authorized and empowered to enter into and perform this Software License Agreement. If, for any reason, any provision of this Software License Agreement is held invalid or otherwise unenforceable, such invalidity or unenforceability shall not affect the remainder of this Software License Agreement, and this Software License Agreement shall continue in full force and effect to the fullest extent allowed by law. The parties knowingly and expressly consent to the foregoing terms and conditions.

Last updated: 2011-10-01

 © 2012 Altova GmbH

Index

.

.docx (Enterprise Edition only), 19, 33
.NET extension functions,
constructors, 577
datatype conversions, .NET to XPath/XQuery, 580
datatype conversions, XPath/XQuery to .NET, 579
for XSLT and XQuery, 575
in XPath expressions, 441
instance methods, instance fields, 578
overview, 575
static methods, static fields, 577
support for, in Authentic View, 441

A

Abbreviations,
used in user manual, 6
About StyleVision, 550
Activating the software, 549
Active configuration, 537
Adding schema, 445
Additional editing procedures, 346
Alias,
see Global Resources, 347
Aligning table cell content,
in SPSs, 526
Altova Engines,
in Altova products, 591
Altova extension functions,
chart functions (see chart functions), 584
general functions, 584
Altova extensions,
chart functions (see chart functions), 584
Altova Global Resources,
see under Global Resources, 347
Altova website, 550
Altova XML Parser,
about, 590
Altova XSLT 1.0 Engine,
limitations and implementation-specific behavior, 554
Altova XSLT 2.0 Engine,
general information about, 557
information about, 556
AltovaXML, 404, 412
and FOP, 410, 411
Append,
column to table in SPS, 519
row to table in SPS, 519
Appendices, 552
ASP.NET application, 398
ASPX web application, 398
Assign predefined formats,
in Quick Start tutorial, 73
atomization of nodes,
in XPath 2.0 and XQuery 1.0 evaluation, 561
Authentic Browser, 27
Authentic Desktop, 27
Authentic View,
in Altova products, 27
Auto Hide,
feature of Design Entry Helpers, 35
Auto-Calculations,
and conditions, 247
and output escaping, 315
command for inserting in design, 482
creating, editing, formatting, 238
example files, 240
examples, 253
formatting of date results, 382
how to use, 237
in Quick Start tutorial, 79
Java and :NET functions in (Enterprise edition only), 238
moving, 238
symbol in Design View, 437
updating node with value of, 482
Automated processing, 404
Auto-numbering, 294

B

Background Information, 588
backwards compatibility,
of XSLT 2.0 Engine, 557
Base year,
in input formatting, 317

Batch files,
and PDF (Enterprise edition only), 414
and scheduled tasks, 413
and scheduled tasks (in Windows Vista), 418
and scheduled tasks (in Windows XP), 415
creating, 414
for generating files from SPS via command line, 405
Blueprints for layout, 163
Bookmarks, 161, 300
command for inserting in design, 491
creating and editing, 301
deleting, 301
enclosing with, 510
Bookmarks (anchors),
symbol in Design View, 437
Borders,
of SPS tables, 523
Bullets and Numbering, 141, 142, 144, 488, 528
enclosing with, 509
Buttons, 160

C

CALS/HTML tables, 137, 525
Catalog files, 103
CDATA sections, 109
Cell (of table),
split horizontally, 522
split vertically, 522
Cells,
joining in SPS tables, 521
Change To command, 173
character entities,
in HTML output of XSLT transformation, 554
Character references,
and output escaping, 315
Check boxes, 156
Class attributes,
in Quick Start tutorial, 73
Close (SPS) command, 452
collations,
in XPath 2.0, 561
Column,
append to SPS table, 519
delete from table in SPS, 520
insert in SPS table, 519
Columns (of tables),
hiding in HTML output, 135
Combo box,
in Quick Start tutorial, 82
Combo boxes, 157
Command line, 404
and parameters, 263
and scheduled tasks, 413
and scheduled tasks (in Windows Vista), 418
and scheduled tasks (in Windows XP), 415
examples of commands, 409
syntax, 406
using StyleVision from, 405
Command line utility, 26
Commands,
customizing, 538
Companion software,
for download, 550
Complex global template, 209
Component download center,
at Altova web site, 550
Composite styles, 341
Condition,
command for inserting in design, 494
Conditional templates, 494
see under: Conditions, 242
symbol in Design View, 437
Conditions,
and Auto-Calculations, 247
editing, 246
enclosing with, 511
in Quick Start tutorial, 82
setting up, 243
Configurations,
of a global resource, 348, 537
Configurations in global resources, 360
Consecutive markup, 32
Content editing procedures, 108
Contents,
command for inserting in design, 479
Contents placeholder,
in Quick Start tutorial, 62
inserting node as contents, 109
Context node,
in XPath dialog, 441
Copy command, 468
Copyright information, 596
count() function,

count() function,
in XPath 1.0, 554
count() function in XPath 2.0,
see fn:count(), 561
Creating new SPS document,
in Quick Start tutorial, 59
Cross references, 298
CSS files,
managing in Design Overview sidebar, 38
CSS styles,
in Modular SPSs, 198
in Quick Start tutorial, 73
see also Styles, 50
CSS stylesheets,
also see Styles, 328
external stylesheets, 328
import precedence of external, 328
media applied to, 328
Custom dictionaries,
for SPS spell-checks, 533
Customize dialog,
for customizing StyleVision, 475
Customizing StyleVision, 538
Cut command, 468

D

Database,
toolbar buttons for editing, 435
Database (Enterprise and Professional editions),
see under DB, 3
Databases,
and global resources, 359
Data-entry devices, 153
menu commands for inserting, 481
symbol in Design View, 437
datatypes,
in XPath 2.0 and XQuery 1.0, 561
Date,
formatting of, 317
Dates,
examples of data manipulation with XPath 2.0, 381
formatting of, 382
how to use in SPS, 381
DB Parameters,
creating and editing, 472
Decimals,
formatting of, 317
deep-equal() function in XPath 2.0,
see fn:deep-equal(), 561
default functions namespace,
for XPath 2.0 and XQueyr 1.0 expressions, 561
in XSLT 2.0 stylesheets, 557
Default user dictionary,
for SPS spell-checks, 533
Delete,
column from table in SPS, 520
row from table in SPS, 520
table in SPS, 517
Delete command, 468
Design elements, 431
Design Entry Helper windows,
docking, 35
floating, 35
Design Entry Helpers,
Auto Hide, 35
description of, 35
Hide, 35
switching display on and off, 476
Design Filters,
switching on and off, 477
Design Fragment,
insert, 499
Design Fragments, 221
Design Overview,
sidebar window, 38
Design structure, 178
Design Tree,
and Modular SPSs, 198
see also Design Entry Helpers, 35
sidebar window, 44
Design View, 436
and JavaScript Editor, 32
description of, 32
display of markup, 32
symbols in SPS design, 437
Dictionaries,
for SPS spell-checks, 533
disable-output-escaping, 315
Distribution,
of Altova's software products, 596, 597, 599
Docking,
Design Entry Helper windows, 35
Document element,

Document element,
definition of, 22
Document elements (see Root elements), 180
Document node,
definition of, 22
Document properties, 234
Document styles, 234
Document views,
in GUI, 31
Documentation,
overview of, 6
Documents,
opening and closing, 31
DPI, 465
DTD,
declaring unparsed entities, 361
DTDs,
as SPS source, 182
Dynamic content,
in Quick Start tutorial, 62
Dynamic lists, 141, 144, 488
Dynamic table,
toolbar buttons for editing, 429
Dynamic tables, 120
and global templates, 124
difference from appended/inserted rows, 124
headers and footers in, 124
nested dynamic tables, 124
see also SPS tables, 124
see also Tables, 131

E

Edit menu, 468
Edit Parameters dialog, 472
Edit Template Match command, 114
Edit Xpath Expression dialog,
see XPath dialog, 441
Element templates,
user-defined, 118
Elements,
user-defined, 118
Enclose With menu, 504
Encoding,
for output files, 544
Encoding command, 465
Encoding of output documents, 465
End User License Agreement, 596, 600
Engine information, 553
Entities,
unparsed, 361
using as URI holders, 361
Entity references,
and output escaping, 315
Entry helpers in Design View,
switching display on and off, 476
Evaluation key,
for your Altova software, 549
Evaluation period,
of Altova's software products, 596, 597, 599
Event handlers,
assigning functions to, 387
Exit command, 467
Extension functions for XSLT and XQuery, 566
Extension Functions in .NET for XSLT and XQuery,
see under .NET extension functions, 575
Extension Functions in Java for XSLT and XQuery,
see under Java extension functions, 567
Extension Functions in MSXSL scripts, 581

F

FAQs on StyleVision, 550
Features,
of StyleVision, 19
File menu, 444
command Exit, 467
File | Close, 452
File | Encoding, 465
File | New, 445
File | Open, 452
File | Print, 466
File | Print Preview, 466
File | Save As, 462
File | Save Design, 457
File | Save Generated Files, 463
File modification alerts,
in Modular SPSs, 198
Files,
open recently used, 467
Filters,
for viewing templates selectively, 433

© 2012 Altova GmbH

Filters for design templates,
switching on and off, 477
Filters on node-templates, 217
Find command, 470
Find Next command, 470
Floating,
Design Entry Helper Windows, 35
fn:base-uri in XPath 2.0,
support in Altova Engines, 563
fn:collection in XPath 2.0,
support in Altova Engines, 563
fn:count() in XPath 2.0,
and whitespace, 561
fn:current-date in XPath 2.0,
support in Altova Engines, 563
fn:current-dateTime in XPath 2.0,
support in Altova Engines, 563
fn:current-time in XPath 2.0,
support in Altova Engines, 563
fn:data in XPath 2.0,
support in Altova Engines, 563
fn:deep-equal() in XPath 2.0,
and whitespace, 561
fn:id in XPath 2.0,
support in Altova Engines, 563
fn:idref in XPath 2.0,
support in Altova Engines, 563
fn:index-of in XPath 2.0,
support in Altova Engines, 563
fn:in-scope-prefixes in XPath 2.0,
support in Altova Engines, 563
fn:last() in XPath 2.0,
and whitespace, 561
fn:lower-case in XPath 2.0,
support in Altova Engines, 563
fn:normalize-unicode in XPath 2.0,
support in Altova Engines, 563
fn:position() in XPath 2.0,
and whitespace, 561
fn:resolve-uri in XPath 2.0,
support in Altova Engines, 563
fn:static-base-uri in XPath 2.0,
support in Altova Engines, 563
fn:upper-case in XPath 2.0,
support in Altova Engines, 563
FO processor (Enterprise edition),
setting up, 26
FO transformations, 410, 411
Footers,
adding in table, 518
in tables, 131
Form controls,
menu commands for inserting, 481
Format strings,
defining for Input Formatting, 529
Formatting,
also see Presentation, 312
for tables, 131
lists, 428
nodes on insertion, 112
of numeric fields, 317
overview of procedures, 312
predefined HTML formats, 428
text alignment, 428
text properties, 428
toolbar buttons for, 428
Formatting numbers,
in Auto-Numbering, 294
Form-based designs, 162, 445
functions,
in XPath, defined by user, 366
see under XSLT 2.0 functions, 559
XPath 2.0 and XQuery 1.0, 560

G

General usage procedure, 96
Generated files, 102
Global Resources, 347
changing configurations, 360
copying configurations, 354
defining, 348
defining file-type, 350
defining folder-type, 353
dialog, 536
selecting configuration via toolbar, 434
toolbar, 434
using, 355, 356, 359, 360
Global Resources XML File, 348
Global styles,
see under Styles, 331
Global templates, 207, 208, 209
effect on rest-of-contents, 113
in Quick Start tutorial, 89

© 2012 Altova GmbH

Global types,
in templates, 209
Graphics,
overview of use in SPS, 147
see also under Images, 147
Grouping, 248
group-by example (Persons.sps), 251
group-by example (Scores.sps), 253
GUI,
description of, 30
document views in, 31
Main Window of, 31
multiple documents in, 31

H

Headers,
adding in table, 518
in tables, 131
Help,
see Onscreen Help, 548
Help menu, 547
Hide,
feature of Design Entry Helpers, 35
Hide markup, 32
Horizontal line,
command for inserting in design, 486
in Quick Start tutorial, 68
HTML import, 390
creating a new SPS, 391
generating files from SPS, 397
of HTML lists, 395
of HTML tables, 395
schema structure, 393
SPS design, 393
HTML output, 102
and image support, 150
HTML tables, 137, 525
HTML to XML conversion, 390
Hyperlink,
command for inserting in design, 492
Hyperlinks, 161, 300
and unparsed entities, 304
creating and editing, 304
enclosing with, 510
linking to bookmarks, 304
linking to external resources, 304
locating via hyperlinks, 361
removing and deleting, 304
symbol in Design View, 437

I

IE 9,
see under Internet Explorer compatibility, 100
Image,
command for inserting in design, 485
Images,
accessing for output rendering, 148
and unparsed entity URIs, 148
example files, 152
in Quick Start tutorial, 68
locating via unparsed entities, 361
specifying URIs for, 148
supported types, 150
symbol in Design View, 437
implementation-specific behavior,
of XSLT 2.0 functions, 559
implicit timezone,
and XPath 2.0 functions, 561
Import of XSLT templates,
into SPS, 224
Input fields, 155
Input formatting,
defining format strings for, 529
of dates, 382
Insert,
column in SPS table, 519
row in SPS table, 519
Insert menu, 478
Bullets and Numbering, 488
Insert | Auto-Calculation, 482
Insert | Bookmarks, 491
Insert | Condition, 494
Insert | Contents, 479
Insert | Design Fragment, 499
Insert | Horizontal Line, 486
Insert | Hyperlink, 492
Insert | Image, 485
Insert | Paragraph, 484
Insert | Rest of contents, 480
Insert | Special Paragraph, 484

© 2012 Altova GmbH

Inserting design elements via the toolbar, 431
Integer,
formatting of, 317
Interface,
see GUI, 30
Internet Explorer compatibility, 100
Internet usage,
in Altova products, 595

J

Java and .NET functions (Enterprise edition only),
in Auto-Calculations, 238
Java extension functions,
constructors, 571
datatype conversions, Java to Xpath/XQuery, 574
datatype conversions, XPath/XQuery to Java, 573
for XSLT and XQuery, 567
in XPath expressions, 441
instance methods, instance fields, 572
overview, 567
static methods, static fields, 572
support for, in Authentic View, 441
user-defined class files, 568
user-defined JAR files, 570
JavaScript,
see under Scripts, 385
JavaScript Editor, 385, 386
in Design View, 32
Joining cells,
in SPS tables, 521

K

Keyboard shortcuts,
customizing for commands, 538
Key-codes,
for your Altova software, 549

L

last() function,
in XPath 1.0, 554
last() function in XPath 2.0,
see fn:last(), 561
Layout,
of views in the GUI, 35
Layout Box, 500
Layout Boxes, 167
Layout Container, 500
Layout Containers, 163
Layout containers and elements, 431
Layout Modules,
steps for creating, 162
Legal information, 596
License, 600
information about, 596
License metering,
in Altova products, 598
Licenses,
for your Altova software, 549
Line,
in Layout Containers, 500
Links,
see under Hyperlinks, 161, 300
List properties, 528
Lists, 141
enclosing with, 509
imported from HTML document, 395
in Quick Start tutorial, 82
Lists (static and dynamic), 488
Local styles,
see under Styles, 334
Local template, 207, 208

M

Main schema, 208
Main schema (Enterprise Edition only), 41
Main template, 207, 208
definition of, 22
Markup tags in Design View, 32
Memory requirements, 589
Menu,
customizing, 538
Menu bar,
moving, 30
Microsoft Office 2007 (Enterprise Edition only), 19, 33
Modular SPS,

© 2012 Altova GmbH

Modular SPS,
activating and de-activating, 198
adding the SPS module, 198
and CSS styles, 195, 198
and file modification alerts, 198
and module objects, 195
and namespace declarations, 195
and schema sources, 195, 198
and Scripts, 195
and Template XML Files, 195
and Working XML Files, 195
creating, 198
effect of order on precedence, 198
example project, 202
overview, 193
the SPS module to add, 198
working with, 198
Modules,
managing in Design Overview sidebar, 38
msxsl:script, 581
Multiline input fields, 155
Multiple document-outputs, 502
Multiple languages examples, 308
Multiple output-documents, 226
and output previews, 232
linking between, 229
location of fiiles, 232

N

Named templates, 207
namespaces,
adding to the SPS, 41, 97, 102, 180
in the SPS, 41
in XSLT 2.0 stylesheet, 557
overview of, 44
New command, 445
New document templates, 226
and design structure, 228
inserting, 227
URLs of, 229
New features, 10
v2010, 12
v2011, 11
New from XSLT, 363
Node,
changing what it is created as, 173
Node-templates,
and chaining to child templates, 217
and global templates, 217
and XPath filters, 217
operations on, 217
User-Defined, 114
Numbering nodes automatically, 294
Numbers,
formatting of, 317
Numeric fields,
formatting of, 317

O

Office Open XML (Enterprise Edition only), 19, 33
Onscreen help,
index of, 548
searching, 548
table of contents, 548
OOXML (Enterprise Edition only), 19, 33
Open,
recently used files, 467
Open (SPS) command, 452
Ordering Altova software, 549
OS,
for Altova products, 589
Otherwise condition branch, 243
Output encoding, 465
Output escaping, 315
Output files,
generating, 102
using command line to generate, 405
Output Views,
description of, 33

P

Paragraph,
command for inserting in design, 484
enclosing with, 508
Parameters, 262
and Authentic View, 263
and command line, 263

Parameters, 262
creating and editing, 472
for design fragments, 265
for schema sources, 268
general description, 263
in SPS, 263
locating nodes in in multiple documents with, 268
managing in Design Overview sidebar, 38
overview of user-defined parameters, 44
Parser,
built into Altova products, 590
Paste command, 468
PDF output (Enterprise edition), 102
and image supportt, 150
Pixels,
and print media lengths, 465
and screen resolution, 465
Platforms,
for Altova products, 589
position() function,
in XPath 1.0, 554
position() function in XPath 2.0,
see fn:position(), 561
Precedence,
of styles, 47
Predefined format strings,
for input formatting, 529
Predefined formats,
command for inserting in design, 484
on inserting a node, 112
symbol in Design View, 437
Presentation,
also see Formats, Formatting, 312
overview of procedures, 312
Print command, 466
Print Preview command, 466
Problems with preview, 26
Processors,
for download, 550
Product features,
listing of, 19
Project options, 544
Properties,
and property groups, 52
defining, 52
of SPS tables, 429, 524
see also Design Entry Helpers, 35
sidebar window, 52
Properties Entry Helper,
Event group, 387
Properties menu, 527
Bullets and Numbering, 528
Properties of output documents, 234
PXF files,
saving as, 462

Q

QName serialization,
when returned by XPath 2.0 functions, 563
Quick Start tutorial,
Auto-Calculations, 79
class attributes, 73
combo boxes, 82
conditions, 82
contents placeholder, 62
creating new SPS document, 59
CSS styles, 73
dynamic content, 62
generating XSLT stylesheets, 93
global templates, 89
horizontal lines, 68
images, 68
introduction, 58
lists, 82
predefined formats, 73
required files, 58
rest-of-contents, 89
setting up new SPS document, 59
static content, 68
static text, 68
testing Authentic View (Enterprise and Professional editions), 93

R

Radio buttons, 160
Recently used files, 467
Redo command, 469
Registering your Altova software, 549
Replace command (Enterprise and Professional editions), 470
Rest-of-contents, 113

Rest-of-contents, 113
and global templates, 209
command for inserting in design, 480
in Quick Start tutorial, 89
Restore toolbars and windows, 543
Right-to-left writing systems, 594
Root elements, 41
Root elements (aka document elements),
and schema sources, 180
selecting for schema, 180
Row,
append to SPS table, 519
delete from table in SPS, 520
insert in SPS table, 519
Rows (of tables),
expanding/collapsing in HTML output, 135
RTF output (Enterprise edition), 102
and image support, 150

S

Save Design command, 457
Save Generated Files command, 463
Scheduled task,
creating a StyleVisionBatch command as, 413, 415, 418
StyleVisionBatch batch files in, 413, 415, 418
Schema sources, 97, 445
and root elements (document elements), 180
changing sources, 268
managing in Design Overview sidebar, 38
multiple in SPS (Enterprise edition), 180
multiple sources and locating nodes, 268
multiple sources and XPath, 268
overview of, 44
selecting for SPS, 180
sidebar window, 41
Schema Sources window,
see also Design Entry Helpers, 35
Schema tree options, 544
schema-awareness,
of XPath 2.0 and XQuery Engines, 561
Schemas,
as SPS source, 182
user-defined, 187
Scripts,
and JavaScript functions, 385
defining JavaScript functions, 386
in the Design Tree, 385
JavaScript functions as event handlers, 387
overview of, 44
using in an SPS, 385
Scripts in XSLT/XQuery,
see under Extension functions, 566
Scroll buttons,
in Main Window, 31
Select All command, 469
Setting up new SPS document,
in Quick Start tutorial, 59
Setting up StyleVision, 26
Shortcuts,
customizing for keyboard, 538
Show markup, 32
Simple global template, 209
Software product license, 600
Sorting, 256
example files, 259
of groups and within groups, 248, 251, 253
Sorting mechanism, 257
Sort-keys, 257
Sort-keys, 256
Source files for SPS, 97
Special paragraph,
command for inserting in design, 484
enclosing with, 508
Spell-checker,
in StyleVision, 532
Spell-checker options,
for SPSs, 533
Split table cell,
horizontally, 522
vertically, 522
SPS,
and Authentic View (Enterprise and Professional editions), 17
and StyleVision, 17
and XSLT stylesheets, 17
closing, 452
general description of, 17
opening, 452
reloading, 452
SPS design overview, 98
SPS file structure, 178
SPS tables,
see also Dynamic tables, 120

© 2012 Altova GmbH

SPS tables,
see aslo Static tables, 120
Static content,
in Quick Start tutorial, 68
Static lists, 141, 142, 488, 509
Static table,
inserting, 517
inserting in SPS, 429
toolbar buttons for editing, 429
Static tables, 120
see also SPS tables, 122
see also Tables, 131
Static text,
and output escaping, 315
in Quick Start tutorial, 68
Status bar, 475
Structure of SPS design, 178
Style Repository,
and external CSS stylesheets, 328
and global styles, 331
see also Design Entry Helpers, 35
sidebar window, 47
Styles,
and property groups, 50
assigning CSS stylesheets to SPS, 328
cascading order, 326
combining several, 341
CSS rules combined, 341
defining, 50
defining global styles in SPS, 331
defining local styles, 334
from XML data, 338
media for assigned external stylesheets, 328
precedence of, 47
precedence of styles, 331
see also Design Entry Helpers, 35
sidebar window, 50
terminology of, 326
via XPath expressions, 338
working with in StyleVision, 326
Styles of output documents, 234
Stylesheets,
also see under CSS stylesheets, 328
also see under XSLT stylesheets, 328
StyleVision,
introduction, 16
product features, 19
running from the command line, 405
user manual, 3
StyleVision Power Stylesheet,
see under SPS, 3
StyleVisionBatch, 26, 404
command line utility, 405
examples of commands, 409
syntax, 406
Support for StyleVision, 550
Support options, 6
Symbols in Design View,
of Auto-Calculations, 437
of bookmarks (anchors), 437
of conditional templates, 437
of data-entry devices, 437
of hyperlinks, 437
of images, 437
of predefined formats, 437
of XML document content, 437
of XML document nodes, 437

T

Table,
adding headers and footers, 518
append column to, 519
append row to, 519
cell content, 516
delete column from, 520
delete row from, 520
deleting in SPS, 517
editing properties of, 524
headers and footers, 516
insert column in, 519
insert row in, 519
inserting a static table, 517
navigating, 516
show/hide borders in StyleVision, 523
vertical alignment of cell content, 526
Table menu, 516
Table of contents,
see under TOC, 271
Tables,
Close button to hide columns, 135
creating, 487
creating dynamic tables, 124
creating static tables, 122

© 2012 Altova GmbH

Tables,
expanding/collapsing rows, 135
formatting, 131
headers and footers in PDF, 131
hiding empty columns, 135
imported from HTML document, 395
joining cells in, 521
overview, 120
styles for alternate rows, 338
Tables (SPS),
editing of properties, 429
toolbar buttons for editing, 429
Tables in Design View,
enclosing with and removing templates, 129
representation of, 129
Tags,
expanding and collapsing, 473
Technical Information, 588
Technical support for StyleVision, 550
Template,
changing the node match for, 173
enclosing with, 505
inserting, 496
Template filters, 433
Template XML File (Enterprise and Professional editions), 97
definition of, 22
Templates,
enclosing table rows and columns with, 129
removing from around table rows and columns, 129
switching view on and off, 477
tree of, 44
Templates for nodes,
see Node-templates, 217
Temporary output document, 26
Terminology,
used in StyleVision, 22
Text references, 298
TOC,
example, hierarchical and sequential, 290
example, simple, 287
marking items for inclusion, 274
menu commands, 501
overview of usage, 271
TOC Bookmarks, 274
and levels, 279
creating, 279
enclosing with, 513
wizard for, 279
TOC items,
constructing, 286
formatting, 286
TOC Levels, 274, 276
enclosing with, 513
TOC references, 285
TOC template,
creating and editing, 282
formatting, 286
level references in, 284
reflevels in, 284
structuring, 284
TOCrefs,
see under TOC references, 285
Toolbar buttons,
adding and removing, 426
Toolbars, 425
adding/removing icons in, 425
customizing, 475
Formatting toolbar, 428
Insert Design Elements toolbar, 431
moving, 30
positioning in GUI, 425
resetting, 425
Standard toolbar, 435
switching display on and off, 475
switching display on/off, 425
Table toolbar, 429
Tools menu, 531
Type-based templates, 209
Types as processing units,
in global templates, 209

U

Ueser-Defined Elements, 118
Ueser-Defined XML Text Blocks, 119
Undo command, 469
Unicode,
support in Altova products, 593
Unicode support,
in Altova products, 592, 594
unparsed-entity-uri function of XSLT, 361
Updating nodes (Enterprise and Professional editions),
with an Auto-Calculation result, 237

© 2012 Altova GmbH

URIs,
holding in unparsed entities, 361
Usage, 96
User Interface,
see GUI, 30
User manual,
see also Onscreen Help, 548
User reference, 424
User-Defined Elements, 117, 503, 515
User-defined schemas, 187
User-defined template,
enclosing with, 506
inserting, 497
User-Defined Templates, 114
User-Defined Text Blocks, 117, 503
User-defined XPath functions, 366

V

Validator,
in Altova products, 590
Value formatting, 317
Variable template, 216
enclosing with, 507
inserting, 498
Variables, 262, 269
Version 2010 new features, 12
Version 2011 new features, 11
Vertical alignment of table cell content,
in SPSs, 526
Vertical text,
in layout boxes, 167
in table cells, 131
View menu, 474
Views,
layout of in GUI, 35

W

whitespace handling,
and XPath 2.0 functions, 561
whitespace in XML document,
handling by Altova XSLT 2.0 Engine, 557
whitespace nodes in XML document,
and handling by XSLT 1.0 Engine, 554
Window menu, 546
Windows,
support for Altova products, 589
Word 2007 (Enterprise Edition only), 19, 33
WordML (Enterprise Edition only), 19, 33
Working XML File, 41, 97
and Output Views, 33
definition of, 22
print preview, 466
printing, 466

X

XML,
inserting in design, 119
XML data,
inserting in SPS design, 109
merging from multiple sources, 190
XML document content,
symbol in Design View, 437
XML document nodes,
symbol in Design View, 437
XML Parser,
about, 590
XML Schemas and DTDs,
as SPS source, 182
XML tables (Enterprise and Professional editions), 120
XMLSpy, 27
XPath,
locating nodes in multiple documents, 268
XPath 1.0,
and dates, 381
XPath 2.0,
and dates, 381
XPath 2.0 functions,
general information about, 561
implementation information, 560
see under fn: for specific functions, 561
XPath dialog,
description of, 441
XPath expressions,
and styles, 338
XPath filter,
on global templates, 209
XPath filters on node-templates, 217

© 2012 Altova GmbH

XPath functions,
in XPath dialog, 441
user-defined, 366
XPath functions support,
see under fn: for individual functions, 563
XPath operators,
in XPath dialog, 441
XPath version in SPS, 99
XQuery,
Extension functions, 566
XQuery 1.0 functions,
general information about, 561
implementation information, 560
see under fn: for specific functions, 561
XQuery processor,
in Altova products, 591
xs:QName,
also see QName, 563
xsl:preserve-space, 554
xsl:strip-space, 554
XSLT,
engine information, 553
Extension functions, 566
inserting code fragment in design, 119
XSLT 1.0 Engine,
limitations and implementation-specific behavior, 554
XSLT 2.0 Engine,
general information about, 557
information about, 556
XSLT 2.0 functions,
implementation-specific behavior of, 559
see under fn: for specific functions, 559
XSLT 2.0 stylesheet,
namespace declarations in, 557
XSLT files,
generating via command line from SPS, 405
XSLT import, 363
XSLT processors,
in Altova products, 591
XSLT stylesheet preview,
in Output Views, 33
XSLT Templates, 44
importing into SPS, 224
managing in Design Overview sidebar, 38
XSLT to SPS, 363
XSLT transformations, 410, 411, 412
XSLT version,
setting for SPS, 435
XSLT version in SPS, 99
XSLTelements,
inserting as code in design, 118

© 2012 Altova GmbH