

User Manual and Programmers' Reference



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 7,739,292.

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 XMLSpy 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

Welcome to XMLSpy	3
User Manual	6
1 New Features	8
2 Introduction	10
2.1 The Graphical User Interface (GUI)	11
2.1.1 Main Window.....	12
2.1.2 Project Window.....	13
2.1.3 Info Window.....	15
2.1.4 Entry Helpers.....	15
2.1.5 Output Window: Messages.....	16
2.1.6 Output Window: XPath.....	17
2.1.7 Output Window: XSL Outline.....	22
2.1.8 Output Window: Find in Files.....	22
2.1.9 Output Window: Find in Schemas.....	24
2.1.10 Output Window: Find in XBRL.....	24
2.1.11 Output Window: Charts.....	25
2.1.12 Menu Bar, Toolbars, Status Bar.....	27
2.2 The Application Environment	29
2.2.1 Settings, Customization, and Menus.....	29
2.2.2 Tutorials, Projects, Examples.....	31
2.2.3 XMLSpy Features and Help, and Altova Products.....	31
3 XMLSpy Tutorial	33
3.1 XMLSpy Interface	34
3.2 XML Schemas: Basics	35
3.2.1 Creating a New XML Schema File.....	35
3.2.2 Defining Namespaces.....	38
3.2.3 Defining a Content Model.....	39

3.2.4	Adding Elements with Drag-and-Drop.....	43
3.2.5	Configuring the Content Model View.....	44
3.2.6	Completing the Basic Schema.....	46
3.3	XML Schemas: Advanced	50
3.3.1	Working with Complex Types and Simple Types.....	50
3.3.2	Referencing Global Elements.....	57
3.3.3	Attributes and Attribute Enumerations.....	59
3.4	XML Schemas: XMLSpy Features	63
3.4.1	Schema Navigation.....	63
3.4.2	Schema Documentation.....	65
3.5	XML Documents	69
3.5.1	Creating a New XML File.....	69
3.5.2	Specifying the Type of an Element.....	71
3.5.3	Entering Data in Grid View.....	73
3.5.4	Entering Data in Text View.....	73
3.5.5	Validating the Document.....	77
3.5.6	Adding Elements and Attributes.....	81
3.5.7	Editing in Database/Table View.....	83
3.5.8	Modifying the Schema.....	87
3.6	XSLT Transformations	90
3.6.1	Assigning an XSLT File.....	90
3.6.2	Transforming the XML File.....	91
3.6.3	Modifying the XSL File.....	92
3.7	Project Management	94
3.7.1	Benefits of Projects.....	94
3.7.2	Building a Project.....	94
3.8	That's It	96
4	Editing Views	97
4.1	Text View	98
4.1.1	Formatting in Text View.....	98
4.1.2	Displaying the Document.....	100
4.1.3	Editing in Text View.....	103
4.1.4	Entry Helpers in Text View.....	105
4.2	Grid View	107
4.2.1	Editing in Grid View.....	108
4.2.2	Grid View Tables.....	109
4.2.3	Entry Helpers in Grid View.....	113
4.3	Schema View	115
4.3.1	Schema Overview.....	115
4.3.2	Content Model View.....	120
4.3.3	Base Type Modification.....	131
4.3.4	Entry Helpers in Schema View.....	132
4.3.5	Identity Constraints.....	135

4.3.6	Smart Restrictions.....	139
4.3.7	xml:base, xml:id, xml:lang, xml:space.....	144
4.3.8	Back and Forward: Moving through Positions.....	145
4.4	WSDL View	147
4.4.1	Main Window.....	148
4.4.2	Overview Entry Helper.....	152
4.4.3	Details Entry Helper.....	156
4.5	XBRL View	157
4.5.1	Main Window: Elements Tab.....	157
4.5.2	Main Window: Definitions, Presentation, Calculation Tabs.....	161
4.5.3	Entry Helpers in XBRL View.....	164
4.6	Authentic View	168
4.6.1	Overview of the GUI.....	168
4.6.2	Authentic View Toolbar Icons.....	169
4.6.3	Authentic View Main Window.....	171
4.6.4	Authentic View Entry Helpers.....	174
4.6.5	Authentic View Context Menus.....	178
4.7	Browser View	180
4.8	Archive View	181

5 XML 183

5.1	Creating, Opening, and Saving XML Documents	184
5.2	Assigning Schemas and Validating	186
5.3	Editing XML in Text View	188
5.4	Editing XML in Grid View	190
5.5	Editing XML in Authentic View	193
5.6	Entry Helpers for XML Documents	195
5.7	Processing with XSLT and XQuery	197
5.8	Charts	199
5.8.1	Creating a Chart.....	201
5.8.2	Source XPath.....	205
5.8.3	X-Axis Selection.....	208
5.8.4	Y-Axis Selection.....	212
5.8.5	Chart Data.....	216
5.8.6	Chart Settings: Quick Reference.....	217
5.8.7	Defining the Chart Type.....	219
5.8.8	Changing the Appearance of a Chart.....	224
	– General.....	225
	– Type-related Features.....	226
	– Colors.....	229
	– X-Axis.....	231
	– Y-Axis.....	233
	– Z-Axis.....	234
	– 3d Angles.....	234

– Sizes	235
– Fonts	236
5.8.9 Chart Overlays	237
5.8.10 Export	238
5.8.11 Chart Example: Simple	239
5.8.12 Chart Example: Advanced	240
5.8.13 Chart Example: Candlestick	246
5.9 XML Signatures	250
5.9.1 Creating XML Signatures	252
5.9.2 Verifying XML Signatures	255
5.9.3 Working with Certificates	257
5.10 Additional Features	261

6 DTDs and XML Schemas 263

6.1 DTDs	264
6.2 XML Schemas	266
6.3 Schema Subsets	267
6.4 Schema Rules	271
6.4.1 Managing Rule Sets	271
6.4.2 Defining a Rule Set	273
6.5 Catalogs in XMLSpy	278
6.6 Working with SchemaAgent	282
6.6.1 Connecting to SchemaAgent Server	283
6.6.2 Opening Schemas Found in the Search Path	285
6.6.3 Using IIRs	286
6.6.4 Viewing Schemas in SchemaAgent	290
6.6.5 SchemaAgent Validation	290
6.7 Find in Schemas	292
6.7.1 Search Term	293
6.7.2 Components	295
6.7.3 Properties	296
6.7.4 Scope	299
6.7.5 Find and Replace Commands	300
6.7.6 Results and Information	302
6.7.7 Finding and Renaming Globals	302

7 XSLT and XQuery 305

7.1 XSLT	306
7.1.1 XSLT Documents	306
7.1.2 XSLT Processing	308
7.1.3 XSL Outline	310
– XSL Outline Window	311

	– Info Window.....	314
7.2	XQuery	317
7.2.1	XQuery Documents.....	318
7.2.2	XQuery Entry Helpers.....	319
7.2.3	XQuery Syntax Coloring.....	319
7.2.4	XQuery Intelligent Editing.....	321
7.2.5	XQuery Validation and Execution.....	322
7.2.6	XQuery and XML Databases.....	323
7.3	XSLT and XQuery Debugger	327
7.3.1	Mechanism and Interface.....	328
7.3.2	Commands and Toolbar Icons.....	329
7.3.3	XSLT/XQuery Settings.....	332
7.3.4	Starting a Debugging Session.....	334
7.3.5	Information Windows.....	336
	– Context Window.....	337
	– Variables Window.....	337
	– XPath-Watch Window.....	338
	– Call Stack Window.....	338
	– Messages Window.....	339
	– Templates Window.....	339
	– Info Window.....	339
	– Trace Window.....	340
	– Arranging the Info Windows.....	340
7.3.6	Breakpoints.....	341
7.3.7	Tracepoints.....	344
7.4	XSLT and XQuery Profiler	350
7.4.1	XSLT Profiling.....	354
7.4.2	XQuery Profiling.....	357
7.4.3	Profiler Charts.....	360

8 Authentic 363

8.1	Authentic View Tutorial	365
8.1.1	Opening an XML Document in Authentic View.....	366
8.1.2	The Authentic View Interface.....	367
8.1.3	Node Operations.....	369
8.1.4	Entering Data in Authentic View.....	372
8.1.5	Entering Attribute Values.....	374
8.1.6	Adding Entities.....	374
8.1.7	Printing the Document.....	375
8.2	Editing in Authentic View	377
8.2.1	Basic Editing.....	377
8.2.2	Tables in Authentic View.....	381
	– SPS Tables.....	382
	– CALS/HTML Tables.....	383

– CALS/HTML Table Editing Icons.....	387
8.2.3 Editing a DB.....	389
– Navigating a DB Table.....	389
– DB Queries.....	390
– Modifying a DB Table.....	394
8.2.4 Working with Dates.....	395
– Date Picker.....	395
– Text Entry.....	396
8.2.5 Defining Entities.....	396
8.2.6 XML Signatures.....	398
8.2.7 Images in Authentic View.....	399
8.2.8 Keystrokes in Authentic View.....	400
8.3 Authentic Scripting	401
 9 HTML, CSS, JSON	 403
9.1 HTML	404
9.2 CSS	406
9.3 JSON	410
 10 WSDL and SOAP	 413
10.1 WSDL Tutorial	414
10.1.1 Creating a New Document.....	414
10.1.2 Creating a PortType.....	415
10.1.3 Creating a Binding.....	416
10.1.4 Creating a Service and Ports.....	418
10.1.5 Validating the WSDL Document.....	419
10.1.6 Connecting to a Web Service and Opening Files.....	419
10.1.7 Sending a SOAP Request from the WSDL File.....	420
10.1.8 Creating WSDL Documentation.....	421
10.1.9 Converting to WSDL 2.0.....	423
10.2 SOAP	424
10.2.1 SOAP Validation.....	424
10.2.2 SOAP Communications Process.....	425
10.2.3 Using SOAP Debugger.....	426
10.2.4 Breakpoints in SOAP Debugger.....	432
 11 XBRL	 435
11.1 Taxonomies: New and Existing	436
11.2 Taxonomy Files Overview	437
11.3 Creating a New Taxonomy	439
11.4 Namespaces in the Taxonomy	443

11.5	Importing a Taxonomy	445
11.6	Setting Up the Taxonomy Files	447
11.7	Adding Elements to a Taxonomy	450
11.8	Relationships and Linkroles	454
11.9	Creating Relationships: Part 1	456
11.10	Creating Relationships: Part 2	459
11.11	Find in XBRL	462
11.11.1	Search Term.....	462
11.11.2	Command Execution.....	464
11.11.3	Results and Information.....	466
12	Office Open XML, ZIP, EPUB	467
12.1	Working with OOXML Files	469
12.2	OOXML Example Files	471
12.3	ZIP Files	472
12.4	EPUB Files	474
13	Databases	478
13.1	Connecting to a Data Source	479
13.1.1	Connection Wizard.....	480
13.1.2	Existing Connections.....	481
13.1.3	ADO Connections.....	482
13.1.4	ODBC Connections.....	486
13.1.5	JDBC Connections.....	489
13.1.6	Global Resources.....	492
13.2	Supported Databases	494
14	Altova Global Resources	495
14.1	Defining Global Resources	496
14.1.1	Files	498
14.1.2	Folders.....	500
14.1.3	Databases.....	501
14.1.4	Copying Configurations.....	502
14.2	Using Global Resources	504
14.2.1	Assigning Files and Folders.....	504
14.2.2	Assigning Databases.....	507
14.2.3	Changing Configurations	508
15	Projects	509
15.1	Creating and Editing Projects	510

15.2	Using Projects	514
16	File/Directory Comparisons	516
16.1	File Comparisons	517
16.2	Directory Comparisons	518
17	Source Control	519
17.1	Supported Source Control Systems	521
17.2	Installing Source Control Systems	526
17.3	SCSs and Altova DiffDog Differencing	533
18	XMLSpy in Visual Studio	539
18.1	Installing the XMLSpy Plugin	540
18.2	Differences with XMLSpy Standalone	542
18.3	XMLSpy's Debuggers in Visual Studio	544
18.4	Known Issues	545
19	XMLSpy in Eclipse	546
19.1	Installing the XMLSpy Plugin for Eclipse	547
19.2	XMLSpy Entry Points in Eclipse	551
19.3	XMLSpy's Debugger Perspectives	554
20	Code Generator	555
20.1	Introduction to code generator	556
20.2	What's new	558
20.3	Generating source code from a schema	560
20.4	Using the generated code library	562
20.4.1	Example schema	563
20.4.2	Using the generated Java library	564
20.4.3	Using the generated C++ library	570
20.4.4	Using the generated C# library	577
20.4.5	Using generated code compatible to old versions	583
	– Creating XML files (XMLSpy 2006)	584
	– Creating XML files (XMLSpy 2005)	586
	– Opening and parsing existing XML files (XMLSpy 2006)	588
	– Opening and parsing existing XML files (XMLSpy 2005)	591
20.5	Code generation tips	594
20.6	Code generator options	596

20.7	The way to SPL (Spy Programming Language)	597
20.7.1	Basic SPL structure	597
20.7.2	Declarations	598
20.7.3	Variables	599
20.7.4	Predefined variables	600
20.7.5	Creating output files	601
20.7.6	Operators	602
20.7.7	Conditions	602
20.7.8	Collections and foreach	603
20.7.9	Subroutines	604
	– Subroutine declaration	605
	– Subroutine invocation	606
	– Subroutine example	607
20.7.10	Built in Types	608
	– Library	608
	– Namespace	608
	– Type	609
	– Member	609
	– NativeBinding	610
	– Facets	610
	– Old object model (up to v2007)	611
	<i>Namespace</i>	611
	<i>Class</i>	611
	<i>Member</i>	612
	<i>Facet</i>	614
	<i>Enumeration</i>	614
	<i>Pattern</i>	614
20.8	Error Codes	615

21 User Reference 616

21.1	File Menu	617
21.1.1	New	617
21.1.2	Open	621
21.1.3	Reload	626
21.1.4	Encoding	626
21.1.5	Close, Close All, Close All But Active	626
21.1.6	Save, Save As, Save All	627
21.1.7	Send by Mail	632
21.1.8	Print	633
21.1.9	Print Preview, Print Setup	634
21.1.10	Recent Files, Exit	635
21.2	Edit Menu	636
21.2.1	Undo, Redo	636
21.2.2	Cut, Copy, Paste, Delete	637

21.2.3	Copy as XML Text.....	637
21.2.4	Copy as Structured Text.....	638
21.2.5	Copy XPath.....	640
21.2.6	Copy XPointer.....	640
21.2.7	Insert	640
21.2.8	Pretty-Print XML Text.....	644
21.2.9	Select All.....	644
21.2.10	Find, Find Next.....	644
21.2.11	Replace.....	647
21.2.12	Find in Files.....	649
21.2.13	Bookmark Commands.....	651
21.2.14	Comment In/Out.....	651
21.3	Project Menu	653
21.3.1	New Project.....	655
21.3.2	Open Project.....	655
21.3.3	Reload Project.....	656
21.3.4	Close Project.....	656
21.3.5	Save Project, Save Project As.....	656
21.3.6	Source Control.....	656
	– Open from Source Control	657
	– Enable Source Control.....	658
	– Get Latest Version.....	659
	– Get	659
	– Get Folders.....	660
	– Check Out.....	661
	– Check In.....	662
	– Undo Check Out.....	663
	– Add to Source Control.....	664
	– Remove from Source Control.....	664
	– Share from Source Control.....	665
	– Show History.....	666
	– Show Differences.....	667
	– Show Properties.....	668
	– Refresh Status.....	669
	– Source Control Manager.....	669
	– Change Source Control.....	669
21.3.7	Add Files to Project.....	670
21.3.8	Add Global Resource to Project.....	670
21.3.9	Add URL to Project.....	670
21.3.10	Add Active File to Project.....	671
21.3.11	Add Active And Related Files to Project.....	671
21.3.12	Add Project Folder to Project.....	671
21.3.13	Add External Folder to Project.....	671
21.3.14	Add External Web Folder to Project.....	674
21.3.15	Script Settings.....	677

21.3.16	Properties.....	678
21.3.17	Most Recently Used Projects.....	680
21.4	XML Menu	681
21.4.1	Insert	681
	– Insert Attribute.....	682
	– Insert Element.....	682
	– Insert Text.....	682
	– Insert CDATA.....	683
	– Insert Comment.....	683
	– Insert XML.....	683
	– Insert Processing Instruction.....	683
	– Insert XInclude.....	683
	– Insert DOCTYPE.....	685
	– Insert ExternalID.....	686
	– Insert ELEMENT.....	687
	– Insert ATTLIST.....	687
	– Insert ENTITY.....	687
	– Insert NOTATION.....	687
	– Insert Encoded External File.....	687
21.4.2	Append.....	688
	– Append Attribute.....	689
	– Append Element.....	689
	– Append Text.....	689
	– Append CDATA.....	690
	– Append Comment.....	690
	– Append XML.....	690
	– Append Processing Instruction.....	690
	– Append XInclude.....	690
	– Append DOCTYPE.....	693
	– Append ExternalID.....	693
	– Append ELEMENT.....	694
	– Append ATTLIST.....	694
	– Append ENTITY.....	694
	– Append NOTATION.....	694
	– Append Encoded External File.....	694
21.4.3	Add Child.....	695
	– Add Child Attribute.....	696
	– Add Child Element.....	696
	– Add Child Text.....	696
	– Add Child CDATA.....	696
	– Add Child Comment.....	696
	– Add Child XML.....	696
	– Add Child Processing Instruction.....	697
	– Add Child XInclude.....	697
	– Add Child DOCTYPE.....	699

	– Add Child ExternalID.....	699
	– Add Child ELEMENT.....	699
	– Add Child ATTLIST.....	700
	– Add Child ENTITY.....	700
	– Add Child NOTATION.....	700
	– Add Child Encoded External File.....	700
21.4.4	Convert To.....	701
	– Convert To Attribute.....	701
	– Convert To Element.....	701
	– Convert To Text.....	701
	– Convert To CDATA.....	701
	– Convert To Comment.....	702
	– Convert To XML.....	702
	– Convert To Processing Instruction.....	702
	– Convert To DOCTYPE.....	702
	– Convert To ExternalID.....	702
	– Convert To ELEMENT.....	702
	– Convert To ATTLIST.....	702
	– Convert To ENTITY.....	702
	– Convert To NOTATION.....	703
21.4.5	Table	703
	– Display as Table.....	703
	– Insert Row.....	704
	– Append Row.....	704
	– Ascending Sort.....	704
	– Descending Sort.....	705
21.4.6	Move Left.....	705
21.4.7	Move Right.....	705
21.4.8	Enclose in Element.....	706
21.4.9	Evaluate XPath.....	706
21.4.10	Check Well-Formedness.....	706
21.4.11	Validate XML.....	707
21.4.12	Validating WSDL Files.....	710
21.4.13	Update Entry Helpers.....	711
21.4.14	Namespace Prefix.....	712
21.4.15	Create XML Signature.....	712
21.4.16	Verify XML Signature.....	715
21.5	DTD/Schema Menu	718
21.5.1	Assign DTD.....	718
21.5.2	Assign Schema.....	719
21.5.3	Include Another DTD.....	719
21.5.4	Go to DTD.....	719
21.5.5	Go to Schema.....	719
21.5.6	Go to Definition.....	720
21.5.7	Generate DTD/Schema.....	720

21.5.8	Convert DTD/Schema.....	722
21.5.9	Convert to UML.....	724
21.5.10	Generate XML from DB, Excel, EDI with MapForce.....	726
21.5.11	Design HTML/PDF/Word Output with StyleVision.....	726
21.5.12	Generate Sample XML File.....	726
21.5.13	Generate Program Code.....	727
21.5.14	Flush Memory Cache.....	728
21.6	Schema Design Menu	729
21.6.1	Schema Settings.....	729
21.6.2	Save Diagram.....	731
21.6.3	Generate Documentation.....	731
	– Documentation Options.....	732
	– User-Defined Design.....	734
21.6.4	Configure View.....	736
21.6.5	Zoom	739
21.6.6	Display All Globals.....	740
21.6.7	Display Diagram.....	740
21.6.8	Schema Extensions for Databases.....	740
	– Enable Oracle Schema Extensions	740
	– Oracle Schema Settings.....	741
	– Enable Microsoft SQL Server Schema Extensions.....	741
	– Named Schema Relationships	742
	– Unnamed Element Relationships.....	743
21.6.9	Connect to SchemaAgent Server.....	743
21.6.10	Disconnect from SchemaAgent Server.....	744
21.6.11	Show in SchemaAgent.....	744
21.6.12	SchemaAgent Validation.....	745
21.6.13	Create Schema Subset.....	745
21.6.14	Flatten Schema.....	746
21.7	XSL/XQuery Menu	748
21.7.1	XSL Transformation.....	749
21.7.2	XSL-FO Transformation.....	750
21.7.3	XSL Parameters / XQuery Variables.....	751
21.7.4	XQuery Execution.....	755
21.7.5	Enable XSLT/XQuery Profiling.....	755
21.7.6	Assign XSL.....	755
21.7.7	Assign XSL-FO.....	756
21.7.8	Assign Sample XML File.....	756
21.7.9	Go to XSL.....	756
21.7.10	Start Debugger / Go.....	756
21.7.11	Stop Debugger.....	757
21.7.12	Restart Debugger.....	757
21.7.13	End Debugger Session.....	757
21.7.14	Step Into.....	757
21.7.15	Step Out.....	757

21.7.16	Step Over.....	757
21.7.17	Show Current Execution Node.....	758
21.7.18	Insert/Remove Breakpoint.....	758
21.7.19	Insert/Remove Tracepoint.....	758
21.7.20	Enable/Disable Breakpoint.....	758
21.7.21	Enable/Disable Tracepoint.....	759
21.7.22	Breakpoints/Tracepoints.....	759
21.7.23	Debug Windows.....	760
21.7.24	XSLT/XQuery Settings.....	760
21.8	Authentic Menu	761
21.8.1	New Document.....	761
21.8.2	Edit Database Data.....	762
21.8.3	Assign/Edit a StyleVision Stylesheet.....	763
21.8.4	Select New Row with XML Data for Editing.....	764
21.8.5	XML Signature.....	765
21.8.6	Define XML Entities.....	766
21.8.7	View Markup.....	767
21.8.8	RichEdit.....	768
21.8.9	Append/Insert/Duplicate/Delete Row.....	768
21.8.10	Move Row Up/Down.....	769
21.8.11	Trusted Locations.....	769
21.9	DB Menu	771
21.9.1	Connecting to a Data Source.....	771
	– Connection Wizard.....	772
	– Existing Connections.....	774
	– ADO Connections.....	775
	– ODBC Connections.....	779
	– JDBC Connections.....	782
	– Global Resources.....	785
21.9.2	Query Database.....	786
	– Data Sources.....	788
	– Browser Pane: Viewing the DB Objects.....	789
	– Query Pane: Description and Features.....	793
	– Query Pane: Working with Queries.....	796
	– Results and Messages.....	797
21.9.3	IBM DB2.....	799
	– Manage XML Schemas.....	799
	– Assign XML Schema.....	802
21.9.4	SQL Server.....	804
	– Manage XML Schemas.....	804
21.9.5	Oracle XML DB.....	806
	– Manage XML Schemas.....	807
	– Browse Oracle XML documents.....	809
21.10	Convert Menu	811
21.10.1	Import Text File.....	811

21.10.2	Import Database Data	813
21.10.3	Import Microsoft Word Document.....	817
21.10.4	Create XML Schema from DB Structure.....	818
21.10.5	DB Import Based on XML Schema	822
21.10.6	Create DB Structure from XML Schema.....	823
21.10.7	Export to Text Files.....	826
21.10.8	Export to a Database.....	829
21.10.9	Convert XML to/from JSON.....	831
21.11	View Menu	833
21.11.1	Text View.....	833
21.11.2	Enhanced Grid View.....	833
21.11.3	Schema Design View.....	833
21.11.4	WSDL Design View.....	834
21.11.5	XBRL Taxonomy View.....	834
21.11.6	Authentic View.....	834
21.11.7	Browser View	834
21.11.8	Expand.....	834
21.11.9	Collapse.....	835
21.11.10	Expand Fully.....	835
21.11.11	Collapse Unselected.....	835
21.11.12	Optimal Widths.....	835
21.11.13	Word Wrap.....	835
21.11.14	Go to Line/Character.....	836
21.11.15	Go to File.....	836
21.11.16	Text View Settings.....	836
21.12	Browser Menu	838
21.12.1	Back	838
21.12.2	Forward.....	838
21.12.3	Stop	838
21.12.4	Refresh.....	838
21.12.5	Fonts	838
21.12.6	Separate Window.....	838
21.13	WSDL Menu	840
21.13.1	WSDL 1.1 Components.....	840
	– Messages.....	840
	– Operations.....	841
	– PortType.....	841
	– Binding.....	841
	– Service.....	842
21.13.2	WSDL 2.0 Components.....	842
	– Interface.....	842
	– Binding.....	844
	– Service.....	844
21.13.3	Types, Save Diagram.....	845
21.13.4	Generate Documentation.....	845

– Documentation Options.....	846
– User-Defined Design.....	848
21.13.5 Reparse WSDL Document.....	849
21.13.6 Convert to WSDL 2.0.....	849
21.13.7 Generate WSDL Program Code with MapForce.....	850
21.14 SOAP Menu	851
21.14.1 Create New SOAP Request.....	851
21.14.2 Send Request to Server.....	853
21.14.3 Change SOAP Request Parameters.....	854
21.14.4 Soap Debugger Session.....	855
21.14.5 Go	856
21.14.6 Single Step.....	856
21.14.7 Break on Next Request.....	856
21.14.8 Break on Next Response.....	857
21.14.9 Stop the Proxy Server	857
21.14.10 Soap Debugger Options.....	857
21.15 XBRL Menu	858
21.15.1 Arcroles.....	858
21.15.2 Linkroles.....	860
21.15.3 Namespace Prefixes.....	861
21.15.4 Set Target Namespace.....	862
21.15.5 Import/Reference.....	863
21.15.6 Generate Documentation.....	864
– Documentation Options.....	864
– User-Defined Design.....	866
21.15.7 View Settings.....	867
21.15.8 Generate XBRL from DB, Excel, CSV with MapForce.....	868
21.15.9 Present XBRL as HTML/PDF/Word with StyleVision.....	868
21.16 Tools Menu	869
21.16.1 Spelling.....	869
21.16.2 Spelling Options.....	873
21.16.3 Scripting Editor.....	877
21.16.4 Macros.....	877
21.16.5 Comparisons.....	878
– Compare Open File With.....	878
– Compare Directories.....	880
– Compare Options.....	883
21.16.6 Global Resources.....	885
21.16.7 Active Configuration.....	885
21.16.8 Customize.....	886
– Commands.....	886
– Toolbars.....	887
– Keyboard.....	888
– Menu.....	893
– Macros.....	895

– Plug-Ins.....	896
– Options.....	897
– Customize Context Menu.....	898
21.16.9 Restore Toolbars and Windows.....	900
21.16.10 Options.....	900
– File	900
– File Types.....	902
– Editing.....	903
– View.....	904
– Grid Fonts.....	906
– Schema Fonts.....	907
– WSDL Fonts.....	907
– XBRL Fonts.....	908
– Text Fonts.....	909
– Colors.....	910
– Encoding.....	912
– XSL.....	912
– Scripting.....	914
– Source Control.....	915
21.17 Window Menu	917
21.17.1 Cascade.....	917
21.17.2 Tile Horizontally.....	917
21.17.3 Tile Vertically.....	917
21.17.4 Project Window.....	917
21.17.5 Info Window.....	917
21.17.6 Entry Helpers.....	918
21.17.7 Output Windows.....	918
21.17.8 Project and Entry Helpers.....	918
21.17.9 All On/Off.....	918
21.17.10 Currently Open Window List.....	918
21.18 Help Menu	920
21.18.1 Table of Contents.....	920
21.18.2 Index	920
21.18.3 Search	920
21.18.4 Keyboard Map.....	921
21.18.5 Activation, Order Form, Registration, Updates.....	921
21.18.6 Support Center, FAQ, Downloads.....	922
21.18.7 On the Internet.....	923
21.18.8 About	923
21.19 Command Line	924

1	Scripting Editor	928
1.1	Overview	930
1.1.1	Scripting Projects in XMLSpy	931
1.1.2	The Scripting Editor GUI	932
1.1.3	Components of a Scripting Project	936
1.2	Creating a Scripting Project	938
1.3	Global Declarations	940
1.4	Forms	942
1.4.1	Creating a New Form	942
1.4.2	Form Design and Form Objects	943
1.4.3	Form Events	945
1.5	Events	948
1.6	Macros	951
1.6.1	Creating and Editing a Macro	951
1.6.2	Running a Macro	953
1.6.3	Debugging a Macro	955
1.7	Programming Points	956
1.7.1	Built-in Commands	957
	– Form usage and commands	964
1.8	Migrating to Scripting Editor 2010 and Later	965
2	IDE Plugins	968
2.1	Registration of IDE PlugIns	969
2.2	ActiveX Controls	970
2.3	Configuration XML	971
2.4	ATL sample files	974
2.4.1	Interface description (IDL)	974
2.4.2	Class definition	976
2.4.3	Implementation	976
2.5	IXMLSpyPlugIn	979
2.5.1	OnCommand	979
2.5.2	OnUpdateCommand	980
2.5.3	OnEvent	980
2.5.4	GetUIModifications	983
2.5.5	GetDescription	983
3	Application API	985
3.1	Overview	987
3.1.1	Object Model	987
3.1.2	Programming Languages	988

– JScript.....	989
<i>Start Application</i>	990
<i>Simple Document Access</i>	990
<i>Iteration</i>	992
<i>Error Handling</i>	993
<i>Events</i>	994
<i>Import and Export of Data</i>	995
<i>Import from Database</i>	998
<i>Export to Database</i>	999
<i>Import from Text</i>	1001
<i>Export to Text</i>	1002
<i>Example: Bubble Sort Dynamic Tables</i>	1004
– VBScript.....	1005
<i>Events</i>	1005
<i>Example: Using Events</i>	1006
– C#	1007
<i>Add Reference to XMLSpy Application API</i>	1011
<i>Application Startup and Shutdown</i>	1012
<i>Opening Documents</i>	1013
<i>Iterating through Open Documents</i>	1014
<i>Errors and COM Output Parameters</i>	1014
<i>Events</i>	1015
– Java.....	1016
<i>Example Java Project</i>	1017
<i>Application Startup and Shutdown</i>	1020
<i>Simple Document Access</i>	1021
<i>Iterations</i>	1022
<i>Use of Out-Parameters</i>	1022
<i>Event Handlers</i>	1023
3.1.3 The DOM and XMLData.....	1023
3.1.4 Obsolete: Authentic View Row operations.....	1025
3.2 Interfaces	1027
3.2.1 Application.....	1027
– Events.....	1029
<i>OnBeforeOpenDocument</i>	1029
<i>OnBeforeOpenProject</i>	1029
<i>OnDocumentOpened</i>	1030
<i>OnProjectOpened</i>	1030
– ActiveDocument.....	1031
– AddMacroMenuItem.....	1031
– Application.....	1031
– ClearMacroMenu.....	1031
– CreateXMLSchemaFromDBStructure.....	1032
– CurrentProject.....	1032
– Dialogs.....	1032

– Documents	1033
– Edition	1033
– FindInFiles	1033
– GetDatabaseImportElementList	1033
– GetDatabaseSettings	1034
– GetDatabaseTables	1034
– GetExportSettings	1035
– GetTextImportElementList	1035
– GetTextImportExportSettings	1036
– ImportFromDatabase	1036
– ImportFromSchema	1037
– ImportFromText	1038
– ImportFromWord	1039
– IsAPISupported	1039
– MajorVersion	1039
– MinorVersion	1039
– NewProject	1040
– OpenProject	1040
– Parent	1040
– Quit	1041
– ReloadSettings	1041
– RunMacro	1041
– ScriptingEnvironment	1042
– ServicePackVersion	1042
– ShowApplication	1042
– ShowFindInFiles	1042
– ShowForm	1043
– Status	1043
– URLDelete	1043
– URLMakeDirectory	1044
– Visible	1044
– WarningNumber	1044
– WarningText	1044
3.2.2 AuthenticContextMenu	1045
– CountItems	1045
– DeleteItem	1045
– GetItemText	1045
– InsertItem	1045
– SetItemText	1046
3.2.3 AuthenticDataTransfer	1046
– dropEffect	1047
– getData	1047
– ownDrag	1047
– type	1047
3.2.4 AuthenticEventContext	1048

	– EvaluateXPath.....	1048
	– GetEventContextType.....	1048
	– GetNormalizedTextValue.....	1049
	– GetVariableValue.....	1049
	– GetXMLNode.....	1049
	– IsAvailable.....	1049
	– SetVariableValue.....	1050
3.2.5	AuthenticRange.....	1050
	– AppendRow.....	1052
	– Application.....	1052
	– CanPerformAction.....	1052
	– CanPerformActionWith.....	1053
	– Clone.....	1053
	– CollapsToBegin.....	1054
	– CollapsToEnd.....	1054
	– Copy.....	1054
	– Cut.....	1054
	– Delete.....	1055
	– DeleteRow.....	1055
	– DuplicateRow.....	1055
	– EvaluateXPath.....	1056
	– ExpandTo.....	1056
	– FirstTextPosition.....	1057
	– FirstXMLData.....	1057
	– FirstXMLDataOffset.....	1058
	– GetElementAttributeNames.....	1059
	– GetElementAttributeValue.....	1059
	– GetElementHierarchy.....	1060
	– GetEntityNames.....	1061
	– GetVariableValue.....	1061
	– Goto.....	1061
	– GotoNext.....	1062
	– GotoNextCursorPosition.....	1062
	– GotoPrevious.....	1063
	– GotoPreviousCursorPosition.....	1063
	– HasElementAttribute.....	1064
	– InsertEntity.....	1064
	– InsertRow.....	1065
	– IsCopyEnabled.....	1065
	– IsCutEnabled.....	1065
	– IsDeleteEnabled.....	1066
	– IsEmpty.....	1066
	– IsEqual.....	1066
	– IsFirstRow.....	1066
	– IsInDynamicTable.....	1067

– IsLastRow.....	1067
– IsPasteEnabled.....	1067
– IsSelected.....	1067
– IsTextStateApplied.....	1068
– LastTextPosition.....	1068
– LastXMLData.....	1069
– LastXMLDataOffset.....	1069
– MoveBegin.....	1070
– MoveEnd.....	1071
– MoveRowDown.....	1071
– MoveRowUp.....	1071
– Parent.....	1072
– Paste.....	1072
– PerformAction.....	1072
– Select.....	1073
– SelectNext.....	1073
– SelectPrevious.....	1074
– SetElementAttributeValue.....	1075
– SetFromRange.....	1076
– SetVariableValue.....	1076
– Text.....	1076
3.2.6 AuthenticView.....	1077
– Events.....	1078
<i>OnBeforeCopy</i>	1078
<i>OnBeforeCut</i>	1078
<i>OnBeforeDelete</i>	1079
<i>OnBeforeDrop</i>	1079
<i>OnBeforePaste</i>	1080
<i>OnBeforeSave</i>	1081
<i>OnDragOver</i>	1081
<i>OnKeyboardEvent</i>	1082
<i>OnLoad</i>	1083
<i>OnMouseEvent</i>	1083
<i>OnSelectionChanged</i>	1084
<i>OnToolbarButtonClicked</i>	1084
<i>OnToolbarButtonExecuted</i>	1086
<i>OnUserAddedXMLNode</i>	1086
– Application.....	1086
– AsXMLString.....	1087
– ContextMenu.....	1087
– CreateXMLNode.....	1087
– DisableAttributeEntryHelper.....	1087
– DisableElementEntryHelper.....	1088
– DisableEntityEntryHelper.....	1088
– DocumentBegin.....	1088

– DocumentEnd.....	1088
– DoNotPerformStandardAction.....	1088
– EvaluateXPath.....	1089
– Event.....	1089
– EventContext.....	1089
– GetToolBarButtonState.....	1089
– Goto.....	1090
– IsRedoEnabled.....	1091
– IsUndoEnabled.....	1091
– MarkupVisibility.....	1091
– Parent.....	1091
– Print.....	1092
– Redo.....	1092
– Selection.....	1092
– SetToolBarButtonState.....	1093
– Undo.....	1093
– UpdateXMLInstanceEntities.....	1093
– WholeDocument.....	1094
– XMLDataRoot.....	1094
3.2.7 CodeGeneratorDlg.....	1094
– Application.....	1095
– CompatibilityMode.....	1096
– CPPSettings_DOMType.....	1096
– CPPSettings_GenerateVC6ProjectFile.....	1096
– CPPSettings_GenerateGCCMakefile.....	1096
– CPPSettings_GenerateVSProjectFile.....	1097
– CPPSettings_LibraryType.....	1097
– CPPSettings_UseMFC.....	1097
– CSharpSettings_ProjectType.....	1098
– OutputPath.....	1098
– OutputPathDialogAction.....	1098
– OutputResultDialogAction.....	1098
– Parent.....	1099
– ProgrammingLanguage.....	1099
– PropertySheetDialogAction.....	1099
– TemplateFileName.....	1100
3.2.8 DatabaseConnection.....	1100
– ADOConnection.....	1101
– AsAttributes.....	1101
– CommentIncluded.....	1101
– CreateMissingTables.....	1102
– CreateNew.....	1102
– DatabaseKind.....	1102
– DatabaseSchema.....	1102
– ExcludeKeys.....	1103

– File.....	1103
– ForeignKeys.....	1103
– ImportColumnsType.....	1103
– IncludeEmptyElements.....	1104
– NullReplacement.....	1104
– NumberDateTimeFormat.....	1104
– ODBCConnection.....	1104
– PrimaryKeys.....	1105
– SchemaExtensionType.....	1105
– SchemaFormat.....	1105
– SQLSelect.....	1105
– TextFieldLen.....	1106
– UniqueKeys.....	1106
3.2.9 Dialogs.....	1106
– Application.....	1107
– CodeGeneratorDlg.....	1107
– FileSelectionDlg.....	1107
– Parent.....	1107
– SchemaDocumentationDlg.....	1108
– GenerateSampleXMLDlg.....	1108
– DTDSchemaGeneratorDlg.....	1108
– FindInFilesDlg.....	1108
– WSDLDocumentationDlg.....	1109
– WSDL20DocumentationDlg.....	1109
– XBRLDocumentationDlg.....	1109
3.2.10 Document.....	1109
– Events.....	1111
<i>OnBeforeSaveDocument</i>	1111
<i>OnBeforeCloseDocument</i>	1112
<i>OnBeforeValidate</i>	1112
<i>OnCloseDocument</i>	1113
<i>OnViewActivation</i>	1113
– Application.....	1114
– AssignDTD.....	1114
– AssignSchema.....	1114
– AssignXSL.....	1115
– AssignXSLFO.....	1115
– AsXMLString.....	1115
– AuthenticView.....	1116
– Close.....	1116
– ConvertDTDOrSchema.....	1117
– ConvertDTDOrSchemaEx.....	1117
– ConvertToWSDL20.....	1117
– CreateChild.....	1118
– CreateDBStructureFromXMLSchema.....	1118

- CreateSchemaDiagram.....	1119
- CurrentViewMode.....	1119
- DataRoot.....	1119
- DocEditView.....	1120
- Encoding.....	1120
- EndChanges.....	1120
- ExecuteXQuery.....	1121
- ExportToDatabase.....	1121
- ExportToText.....	1122
- FullName.....	1123
- GenerateDTDOrSchema.....	1123
- GenerateDTDOrSchemaEx.....	1124
- GenerateProgramCode.....	1124
- GenerateSampleXML.....	1124
- GenerateSchemaDocumentation.....	1124
- GenerateWSDL20Documentation.....	1125
- GenerateWSDLDocumentation.....	1125
- GenerateXBRLDocumentation.....	1126
- GetDBStructureList.....	1126
- GetExportElementList.....	1126
- GetPathName (obsolete).....	1127
- GridView.....	1127
- IsModified.....	1128
- IsValid.....	1128
- IsWellFormed.....	1129
- Name.....	1130
- Parent.....	1130
- Path.....	1130
- RootElement.....	1130
- Save.....	1131
- SaveAs.....	1131
- Saved.....	1131
- SaveInString.....	1131
- SaveToURL.....	1132
- SetActiveDocument.....	1132
- SetEncoding (obsolete).....	1133
- SetExternalIsValid.....	1134
- SetPathName (obsolete).....	1134
- StartChanges.....	1134
- Suggestions.....	1135
- SwitchViewMode.....	1135
- TextView.....	1135
- Title.....	1135
- TransformXSL.....	1136
- TransformXSLEx.....	1136

	– TransformXSLFO.....	1136
	– TreatXBRLInconsistencies AsErrors.....	1136
	– UpdateViews.....	1137
	– UpdateXMLData.....	1137
3.2.11	Documents.....	1137
	– Count.....	1138
	– Item.....	1138
	– NewAuthenticFile.....	1139
	– NewFile.....	1139
	– NewFileFromText.....	1139
	– OpenAuthenticFile.....	1140
	– OpenFile.....	1140
	– OpenURL.....	1140
	– OpenURLDialog.....	1141
3.2.12	DTDSchemaGeneratorDlg.....	1141
	– Application.....	1142
	– AttributeTypeDefinition.....	1142
	– DTDSchemaFormat.....	1142
	– FrequentElements.....	1143
	– GlobalAttributes.....	1143
	– MaxEnumLength.....	1143
	– MergeAllEqualNamed.....	1143
	– OnlyStringEnums.....	1144
	– OutputPath.....	1144
	– OutputPathDialogAction.....	1144
	– Parent.....	1144
	– ResolveEntities.....	1144
	– TypeDetection.....	1145
	– ValueList.....	1145
3.2.13	ElementList.....	1145
	– Count.....	1146
	– Item.....	1146
	– RemoveElement.....	1146
3.2.14	ElementListItem.....	1146
	– ElementKind.....	1146
	– FieldCount.....	1147
	– Name.....	1147
	– RecordCount.....	1147
3.2.15	ExportSettings.....	1147
	– CreateKeys.....	1148
	– ElementList.....	1148
	– EntitiesToText.....	1148
	– ExportAllElements.....	1148
	– ExportCompleteXML.....	1148
	– FromAttributes.....	1149

	– FromSingleSubElements	1149
	– FromTextValues	1149
	– IndependentPrimaryKey	1149
	– Namespace	1149
	– StartFromElement	1149
	– SubLevelLimit	1150
3.2.16	FileSelectionDlg	1150
	– Application	1150
	– DialogAction	1151
	– FullName	1151
	– Parent	1151
3.2.17	FindInFilesDlg	1151
	– AdvancedXMLSearch	1152
	– Application	1152
	– DoReplace	1152
	– FileExtension	1153
	– Find	1153
	– IncludeSubfolders	1153
	– MatchCase	1153
	– MatchWholeWord	1153
	– Parent	1154
	– RegularExpression	1154
	– Replace	1154
	– ReplaceOnDisk	1154
	– SearchInProjectFilesDoExternal	1154
	– SearchLocation	1155
	– ShowResult	1155
	– StartFolder	1155
	– XMLAttributeContents	1155
	– XMLAttributeNames	1155
	– XMLCDATA	1156
	– XMLComments	1156
	– XMLElementContents	1156
	– XMLElementNames	1156
	– XMLPI	1157
	– XMLRest	1157
3.2.18	FindInFilesResult	1157
	– Application	1157
	– Count	1158
	– Document	1158
	– Item	1158
	– Parent	1158
	– Path	1158
3.2.19	FindInFilesResultMatch	1158
	– Application	1159

	– Length.....	1159
	– Line.....	1159
	– LineText.....	1159
	– Parent.....	1160
	– Position.....	1160
	– Replaced.....	1160
3.2.20	FindInFilesResults	1160
	– Application.....	1161
	– Count.....	1161
	– Item.....	1161
	– Parent.....	1161
3.2.21	GenerateSampleXMLDlg	1161
	– Application.....	1162
	– Parent.....	1162
	– NonMandatoryAttributes.....	1162
	– NonMandatoryElements - obsolete.....	1162
	– TakeFirstChoice - obsolete.....	1163
	– RepeatCount.....	1163
	– FillWithSampleData - obsolete.....	1163
	– FillElementsWithSampleData	1163
	– FillAttributesWithSampleData	1163
	– ContentOfNillableElementsIsNonMandatory.....	1164
	– TryToUseNonAbstractTypes.....	1164
	– Optimization.....	1164
	– SchemaOrDTDAssignment	1164
	– LocalNameOfRootElement	1164
	– NamespaceURIOfRootElement.....	1165
	– OptionsDialogAction.....	1165
3.2.22	GridView	1165
	– Events.....	1165
	<i>OnBeforeDrag</i>	1165
	<i>OnBeforeDrop</i>	1166
	<i>OnBeforeStartEditing</i>	1166
	<i>OnEditingFinished</i>	1167
	<i>OnFocusChanged</i>	1167
	– CurrentFocus.....	1168
	– Deselect.....	1168
	– IsVisible.....	1168
	– Select.....	1168
	– SetFocus.....	1168
3.2.23	SchemaDocumentationDlg.....	1169
	– AllDetails	1170
	– Application.....	1170
	– CreateDiagramsFolder.....	1170
	– DiagramFormat.....	1171

– EmbedCSSInHTML.....	1171
– EmbedDiagrams.....	1171
– GenerateRelativeLinks.....	1171
– IncludeAll.....	1172
– IncludeAttributeGroups.....	1172
– IncludeComplexTypes.....	1172
– IncludeGlobalAttributes.....	1173
– IncludeGlobalElements.....	1173
– IncludeGroups.....	1173
– IncludeIndex.....	1173
– IncludeLocalAttributes.....	1174
– IncludeLocalElements.....	1174
– IncludeRedefines.....	1174
– IncludeReferencedSchemas.....	1174
– IncludeSimpleTypes.....	1175
– MultipleOutputFiles.....	1175
– OptionsDialogAction.....	1175
– OutputFile.....	1176
– OutputFileDialogAction.....	1176
– OutputFormat.....	1176
– Parent.....	1177
– ShowAnnotations.....	1177
– ShowAttributes.....	1177
– ShowChildren.....	1177
– ShowDiagram.....	1178
– ShowEnumerations.....	1178
– ShowIdentityConstraints.....	1178
– ShowNamespace.....	1179
– ShowPatterns.....	1179
– ShowProgressBar.....	1179
– ShowProperties.....	1179
– ShowResult.....	1180
– ShowSingleFacets.....	1180
– ShowSourceCode.....	1180
– ShowType.....	1181
– ShowUsedBy.....	1181
– SPSFile.....	1181
– UseFixedDesign.....	1181
3.2.24 SpyProject.....	1182
– CloseProject.....	1182
– ProjectFile.....	1182
– RootItems.....	1183
– SaveProject.....	1183
– SaveProjectAs.....	1183
3.2.25 SpyProjectItem.....	1183

	- ChildItems	1184
	- FileExtensions	1184
	- ItemType	1184
	- Name	1184
	- Open	1184
	- ParentItem	1184
	- Path	1185
	- ValidateWith	1185
	- XMLForXSLTransformation	1185
	- XSLForXMLTransformation	1185
	- XSLTransformationFileExtension	1185
	- XSLTransformationFolder	1186
3.2.26	SpyProjectItems	1186
	- AddFile	1186
	- AddFolder	1186
	- AddURL	1187
	- Count	1187
	- Item	1187
	- RemoveItem	1187
3.2.27	TextImportExportSettings	1188
	- CommentIncluded	1188
	- DestinationFolder	1188
	- EnclosingCharacter	1188
	- Encoding	1189
	- EncodingByteOrder	1189
	- FieldDelimiter	1189
	- FileExtension	1189
	- HeaderRow	1189
	- ImportFile	1190
	- RemoveDelimiter	1190
	- RemoveNewline	1190
3.2.28	TextView	1190
	- Events	1191
	<i>OnBeforeShowSuggestions</i>	1191
	<i>OnChar</i>	1191
	- Application	1192
	- GetRangeText	1192
	- GoToLineChar	1192
	- Length	1192
	- LineCount	1193
	- LineFromPosition	1193
	- LineLength	1193
	- MoveCaret	1193
	- Parent	1193
	- PositionFromLine	1194

	– ReplaceText.....	1194
	– SelectionEnd.....	1194
	– SelectionStart.....	1194
	– SelectText.....	1194
	– SelText.....	1195
	– Text.....	1195
3.2.29	WSDL20DocumentationDlg.....	1195
	– AllDetails.....	1196
	– Application.....	1196
	– CreateDiagramsFolder.....	1197
	– DiagramFormat.....	1197
	– EmbedCSSInHTML.....	1197
	– EmbedDiagrams.....	1197
	– IncludeAll.....	1198
	– IncludeBinding.....	1198
	– IncludeImportedWSDLFiles.....	1198
	– IncludeInterface.....	1199
	– IncludeOverview.....	1199
	– IncludeService.....	1199
	– IncludeTypes.....	1199
	– MultipleOutputFiles.....	1200
	– OptionsDialogAction.....	1200
	– OutputFile.....	1200
	– OutputFileDialogAction.....	1201
	– OutputFormat.....	1201
	– Parent.....	1201
	– ShowBindingDiagram.....	1201
	– ShowEndpoint.....	1202
	– ShowExtensibility.....	1202
	– ShowFault.....	1202
	– ShowInterfaceDiagram.....	1203
	– ShowOperation.....	1203
	– ShowProgressBar.....	1203
	– ShowResult.....	1203
	– ShowServiceDiagram.....	1204
	– ShowSourceCode.....	1204
	– ShowTypesDiagram.....	1204
	– ShowUsedBy.....	1205
	– SPSFile.....	1205
	– UseFixedDesign.....	1205
3.2.30	WSDLDocumentationDlg.....	1205
	– AllDetails.....	1206
	– Application.....	1207
	– CreateDiagramsFolder.....	1207
	– DiagramFormat.....	1207

– EmbedCSSInHTML.....	1207
– EmbedDiagrams.....	1208
– IncludeAll.....	1208
– IncludeBinding.....	1208
– IncludeImportedWSDLFiles.....	1209
– IncludeMessages.....	1209
– IncludeOverview.....	1209
– IncludePortType.....	1209
– IncludeService.....	1210
– IncludeTypes.....	1210
– MultipleOutputFiles.....	1210
– OptionsDialogAction.....	1211
– OutputFile.....	1211
– OutputFileDialogAction.....	1211
– OutputFormat.....	1211
– Parent.....	1212
– ShowBindingDiagram.....	1212
– ShowExtensibility.....	1212
– ShowMessageParts.....	1213
– ShowPort.....	1213
– ShowPortTypeDiagram.....	1213
– ShowPortTypeOperations.....	1213
– ShowProgressBar.....	1214
– ShowResult.....	1214
– ShowServiceDiagram.....	1214
– ShowSourceCode.....	1215
– ShowTypesDiagram.....	1215
– ShowUsedBy.....	1215
– UseFixedDesign.....	1215
– SPSFile.....	1216
3.2.31 XBRLDocumentationDlg.....	1216
– AllDetails.....	1217
– Application.....	1217
– CreateDiagramsFolder.....	1218
– DiagramFormat.....	1218
– EmbedCSSInHTML.....	1218
– EmbedDiagrams.....	1218
– IncludeAll.....	1219
– IncludeCalculationLinkroles.....	1219
– IncludeDefinitionLinkroles.....	1219
– IncludeGlobalElements.....	1220
– IncludeNamespacePrefixes.....	1220
– IncludeOverview.....	1220
– IncludePresentationLinkroles.....	1220
– OptionsDialogAction.....	1221

–	OutputFile.....	1221
–	OutputFileDialogAction.....	1221
–	OutputFormat.....	1222
–	Parent.....	1222
–	ShortQualifiedName.....	1222
–	ShowAbstract.....	1222
–	ShowBalance.....	1223
–	ShowDiagram.....	1223
–	ShowImportedElements.....	1223
–	ShowItemtype.....	1224
–	ShowLabels.....	1224
–	ShowLinkbaseReferences.....	1224
–	ShowNillable.....	1224
–	ShowPeriod.....	1225
–	ShowProgressBar.....	1225
–	ShowReferences.....	1225
–	ShowResult.....	1226
–	ShowSubstitutiongroup.....	1226
–	SPSFile.....	1226
–	UseFixedDesign.....	1226
3.2.32	XMLData.....	1227
–	AppendChild.....	1228
–	CountChildren.....	1228
–	CountChildrenKind.....	1229
–	EraseAllChildren.....	1229
–	EraseChild.....	1229
–	EraseCurrentChild.....	1230
–	GetChild.....	1230
–	GetChildAttribute.....	1231
–	GetChildElement.....	1231
–	GetChildKind.....	1231
–	GetCurrentChild.....	1232
–	GetFirstChild.....	1232
–	GetNamespacePrefixForURI.....	1232
–	GetNextChild.....	1233
–	GetTextValueXMLDecoded.....	1234
–	HasChildren.....	1234
–	HasChildrenKind.....	1234
–	InsertChild.....	1234
–	InsertChildAfter.....	1235
–	InsertChildBefore.....	1235
–	IsSameNode.....	1236
–	Kind.....	1236
–	MayHaveChildren.....	1236
–	Name.....	1236

– Parent.....	1237
– SetTextValueXMLEncoded.....	1237
– TextValue.....	1237
3.3 Interfaces (obsolete)	1238
3.3.1 AuthenticEvent (obsolete).....	1238
– altKey (obsolete).....	1239
– altLeft (obsolete).....	1240
– button (obsolete).....	1241
– cancelBubble (obsolete).....	1242
– clientX (obsolete).....	1243
– clientY (obsolete).....	1244
– ctrlKey (obsolete).....	1245
– ctrlLeft (obsolete).....	1246
– dataTransfer (obsolete).....	1247
– fromElement (obsolete).....	1248
– keyCode (obsolete).....	1248
– propertyName (obsolete).....	1249
– repeat (obsolete).....	1249
– returnValue (obsolete).....	1249
– shiftKey (obsolete).....	1250
– shiftLeft (obsolete).....	1251
– srcElement (obsolete).....	1252
– type (obsolete).....	1253
3.3.2 AuthenticSelection (obsolete).....	1254
– End (obsolete).....	1255
– EndTextPosition (obsolete).....	1255
– Start (obsolete).....	1256
– StartTextPosition (obsolete).....	1256
3.3.3 OldAuthenticView (obsolete).....	1257
– ApplyTextState (obsolete).....	1259
– CurrentSelection (obsolete).....	1260
– EditClear (obsolete).....	1260
– EditCopy (obsolete).....	1261
– EditCut (obsolete).....	1261
– EditPaste (obsolete).....	1262
– EditRedo (obsolete).....	1262
– EditSelectAll (obsolete).....	1263
– EditUndo (obsolete).....	1263
– event (obsolete).....	1264
– GetAllowedElements (obsolete).....	1264
– GetNextVisible (obsolete).....	1267
– GetPreviousVisible (obsolete).....	1268
– IsEditClearEnabled (obsolete).....	1269
– IsEditCopyEnabled (obsolete).....	1269
– IsEditCutEnabled (obsolete).....	1270

– IsEditPasteEnabled (obsolete).....	1270
– IsEditRedoEnabled (obsolete).....	1271
– IsEditUndoEnabled (obsolete).....	1271
– IsRowAppendEnabled (obsolete).....	1272
– IsRowDeleteEnabled (obsolete).....	1272
– IsRowDuplicateEnabled (obsolete).....	1273
– IsRowInsertEnabled (obsolete).....	1273
– IsRowMoveDownEnabled (obsolete).....	1274
– IsRowMoveUpEnabled (obsolete).....	1274
– IsTextStateApplied (obsolete).....	1275
– IsTextStateEnabled (obsolete).....	1275
– LoadXML (obsolete).....	1276
– MarkupView (obsolete).....	1276
– RowAppend (obsolete).....	1278
– RowDelete (obsolete).....	1278
– RowDuplicate (obsolete).....	1279
– RowInsert (obsolete).....	1279
– RowMoveDown (obsolete).....	1280
– RowMoveUp (obsolete).....	1280
– SaveXML (obsolete).....	1281
– SelectionMoveTabOrder (obsolete).....	1282
– SelectionSet (obsolete).....	1283
– XMLRoot (obsolete).....	1284
3.4 Enumerations	1286
3.4.1 ENUMApplicationStatus	1286
3.4.2 SPYAttributeTypeDefinition	1286
3.4.3 SPYAuthenticActions	1286
3.4.4 SPYAuthenticDocumentPosition	1286
3.4.5 SPYAuthenticElementActions	1287
3.4.6 SPYAuthenticElementKind	1287
3.4.7 SPYAuthenticMarkupVisibility	1287
3.4.8 SPYAuthenticToolbarButtonState	1287
3.4.9 SPYDatabaseKind	1288
3.4.10 SPYDialogAction	1288
3.4.11 SPYDOMType	1288
3.4.12 SPYDTDSchemaFormat	1288
3.4.13 SPYEncodingByteOrder	1289
3.4.14 SPYExportNamespace	1289
3.4.15 SPYFindInFilesSearchLocation	1289
3.4.16 SPYFrequentElements	1289
3.4.17 SPYImageKind	1289
3.4.18 SPYImportColumnsType	1289
3.4.19 SPYKeyEvent	1290
3.4.20 SPYKeyStatus	1290
3.4.21 SPYLibType	1290

3.4.22	SPYLoading.....	1290
3.4.23	SPYMouseEvent.....	1290
3.4.24	SPYNumberDateTimeFormat.....	1291
3.4.25	SPYProgrammingLanguage.....	1291
3.4.26	SPYProjectItemTypes.....	1292
3.4.27	SPYProjectType.....	1292
3.4.28	SPYSampleXMLGenerationOptimization.....	1292
3.4.29	SPYSampleXMLGenerationSchemaOrDTDAssignment.....	1292
3.4.30	SPYSchemaDefKind.....	1292
3.4.31	SPYSchemaDocumentationFormat.....	1293
3.4.32	SPYSchemaExtensionType.....	1293
3.4.33	SPYSchemaFormat.....	1293
3.4.34	SPYTextDelimiters.....	1294
3.4.35	SPYTextEnclosing.....	1294
3.4.36	SPYTypeDetection.....	1294
3.4.37	SPYURLTypes.....	1294
3.4.38	SPYViewModes.....	1294
3.4.39	SPYVirtualKeyMask.....	1295
3.4.40	SPYXMLDataKind.....	1295
3.5	Application API for Java	1297
3.5.1	Sample source code.....	1298
3.5.2	SpyApplication.....	1300
3.5.3	SpyCodeGeneratorDlg.....	1301
3.5.4	SpyDatabaseConnection.....	1301
3.5.5	SpyDialogs.....	1302
3.5.6	SpyDoc.....	1302
3.5.7	SpyDocuments.....	1304
3.5.8	SpyDTDSchemaGeneratorDlg.....	1304
3.5.9	SpyElementList.....	1305
3.5.10	SpyElementListItem.....	1305
3.5.11	SpyExportSettings.....	1305
3.5.12	SpyFileSelectionDlg.....	1306
3.5.13	SpyFindInFilesDlg.....	1306
3.5.14	SpyFindInFilesMatch.....	1307
3.5.15	SpyFindInFilesResult.....	1307
3.5.16	SpyFindInFilesResults.....	1307
3.5.17	SpyGenerateSampleXMLDlg.....	1307
3.5.18	SpyGridView.....	1308
3.5.19	SpyProject.....	1308
3.5.20	SpyProjectItem.....	1308
3.5.21	SpyProjectItems.....	1309
3.5.22	SpySchemaDocumentationDlg.....	1309
3.5.23	SpyTextImportExportSettings.....	1311
3.5.24	SpyTextView.....	1311
3.5.25	SpyWSDL20DocumentationDlg.....	1312

3.5.26	SpyWSDLDocumentationDlg.....	1313
3.5.27	SpyXBRLDocumentationDlg.....	1315
3.5.28	SpyXMLData.....	1317
3.5.29	Authentic.....	1317
	– SpyAuthenticRange.....	1317
	– SpyAuthenticView.....	1318
	– SpyDocEditSelection.....	1319
	– SpyDocEditView.....	1319
3.5.30	Predefined constants.....	1320
	– SPYApplicationStatus.....	1320
	– SPYAttributeTypeDefinition.....	1320
	– SPYAuthenticActions.....	1320
	– SPYAuthenticDocumentPosition.....	1321
	– SPYAuthenticElementKind.....	1321
	– SPYAuthenticMarkupVisibility.....	1321
	– SPYDatabaseKind.....	1322
	– SPYDialogAction.....	1322
	– SPYDOMType.....	1322
	– SPYDTDSchemaFormat.....	1322
	– SPYEncodingByteOrder.....	1323
	– SPYExportNamespace.....	1323
	– SPYFindInFilesSearchLocation.....	1323
	– SPYFrequentElements.....	1323
	– SPYImageKind.....	1323
	– SPYImportColumnsType.....	1324
	– SPYLibType.....	1324
	– SPYLoading.....	1324
	– SPYNumberDateTimeFormat.....	1324
	– SPYProgrammingLanguage.....	1324
	– SPYProjectItemTypes.....	1324
	– SPYProjectType.....	1325
	– SPYSampleXMLGenerationOptimization.....	1325
	– SPYSampleXMLGenerationSchemaOrDTDAssignment.....	1325
	– SPYSchemaDefKind.....	1325
	– SPYSchemaDocumentationFormat.....	1326
	– SPYSchemaExtensionType.....	1326
	– SPYSchemaFormat.....	1327
	– SPYTextDelimiters.....	1327
	– SPYTextEnclosing.....	1327
	– SPYTypeDetection.....	1327
	– SPYURLTypes.....	1327
	– SpyViewModes.....	1328
	– SPYWhitespaceComparison.....	1328
	– SPYXMLDataKind.....	1328

4	ActiveX Integration	1330
4.1	Integration at Application Level	1331
4.1.1	Example: HTML.....	1331
	– Instantiate the Control.....	1331
	– Add Button to Open Default Document.....	1331
	– Add Buttons for Code Generation.....	1332
	– Connect to Custom Events.....	1333
4.2	Integration at Document Level	1334
4.2.1	Use XMLSpyControl.....	1334
4.2.2	Use XMLSpyControlDocument.....	1335
4.2.3	Use XMLSpyControlPlaceHolder.....	1335
4.2.4	Query XMLSpy Commands.....	1335
4.2.5	Examples.....	1335
	– C#	1335
	<i>Introduction</i>	1336
	<i>Placing the XMLSpyControl</i>	1336
	<i>Adding the Document Control</i>	1336
	<i>Adding the Placeholder Control</i>	1336
	<i>Retrieving Command Information</i>	1337
	<i>Handling Events</i>	1338
	<i>Testing the Example</i>	1339
	– HTML.....	1340
	<i>Instantiate the XMLSpyControl</i>	1340
	<i>Create Editor Window</i>	1340
	<i>Create Project Window</i>	1340
	<i>Create Placeholder for Helper Windows</i>	1341
	<i>Create a Custom Toolbar</i>	1341
	<i>Create More Buttons</i>	1342
	<i>Create Event Handler to Update Button Status</i>	1343
4.3	Command Table for XMLSpy	1345
4.3.1	File Menu.....	1345
4.3.2	Edit Menu.....	1346
4.3.3	Project Menu.....	1346
4.3.4	XML menu.....	1347
4.3.5	DTD/Schema Menu	1349
4.3.6	Schema Design Menu.....	1349
4.3.7	XSL/XQuery Menu.....	1350
4.3.8	Authentic Menu.....	1351
4.3.9	Convert Menu.....	1352
4.3.10	View Menu	1352
4.3.11	Browser Menu.....	1353
4.3.12	WSDL Menu.....	1353
4.3.13	SOAPMenu.....	1355

4.3.14	Tools Menu.....	1355
4.3.15	Window Menu.....	1356
4.3.16	Help Menu.....	1356
4.4	Accessing XMLSpyAPI	1357
4.5	Object Reference	1358
4.5.1	XMLSpyCommand.....	1358
	– Accelerator.....	1358
	– ID	1358
	– IsSeparator.....	1359
	– Label.....	1359
	– StatusText.....	1359
	– SubCommands.....	1359
	– ToolTip.....	1359
4.5.2	XMLSpyCommands.....	1359
	– Count.....	1360
	– Item.....	1360
4.5.3	XMLSpyControl.....	1360
	– Properties.....	1361
	<i>Appearance</i>	1361
	<i>Application</i>	1361
	<i>BorderStyle</i>	1361
	<i>CommandsList</i>	1361
	<i>EnableUserPrompts</i>	1362
	<i>IntegrationLevel</i>	1362
	<i>MainMenu</i>	1362
	<i>Toolbars</i>	1362
	– Methods.....	1363
	<i>Exec</i>	1363
	<i>Open</i>	1363
	<i>QueryStatus</i>	1363
	– Events.....	1364
	<i>OnCloseEditingWindow</i>	1364
	<i>OnContextChanged</i>	1364
	<i>OnDocumentOpened</i>	1364
	<i>OnFileChangedAlert</i>	1365
	<i>OnLicenseProblem</i>	1365
	<i>OnOpenedOrFocused</i>	1365
	<i>OnToolWindowUpdated</i>	1365
	<i>OnUpdateCmdUI</i>	1366
	<i>OnValidationWindowUpdated</i>	1366
4.5.4	XMLSpyControlDocument.....	1366
	– Properties.....	1367
	<i>Appearance</i>	1367
	<i>BorderStyle</i>	1367
	<i>Document</i>	1367

	<i>IsModified</i>	1367
	<i>Path</i>	1368
	<i>ReadOnly</i>	1368
–	Methods.....	1368
	<i>Exec</i>	1368
	<i>New</i>	1368
	<i>Open</i>	1369
	<i>QueryStatus</i>	1369
	<i>Reload</i>	1369
	<i>Save</i>	1369
	<i>SaveAs</i>	1370
–	Events.....	1370
	<i>OnActivate</i>	1370
	<i>OnContextChanged</i>	1370
	<i>OnDocumentClosed</i>	1371
	<i>OnDocumentOpened</i>	1371
	<i>OnDocumentSaveAs</i>	1371
	<i>OnFileChangedAlert</i>	1371
	<i>OnModifiedFlagChanged</i>	1371
	<i>OnSetEditorTitle</i>	1372
4.5.5	XMLSpyControlPlaceholder.....	1372
–	Properties.....	1372
	<i>Label</i>	1372
	<i>PlaceholderWindowID</i>	1372
	<i>Project</i>	1373
–	Methods.....	1373
	<i>OpenProject</i>	1373
	<i>CloseProject</i>	1373
–	Events.....	1373
	<i>OnModifiedFlagChanged</i>	1374
	<i>OnSetLabel</i>	1374
4.5.6	Enumerations	1374
–	ICActiveXIntegrationLevel.....	1374
–	XMLSpyControlPlaceholderWindow.....	1374

Appendices

1378

1 Engine Information

1379

1.1	XSLT 1.0 Engine: Implementation Information	1380
1.2	XSLT 2.0 Engine: Implementation Information	1382

1.2.1	General Information.....	1382
1.2.2	XSLT 2.0 Elements and Functions.....	1384
1.3	XQuery 1.0 Engine: Implementation Information	1385
1.4	XPath 2.0 and XQuery 1.0 Functions	1388
1.4.1	General Information.....	1388
1.4.2	Functions Support.....	1389
1.5	Extensions	1392
1.5.1	Java Extension Functions.....	1392
	– User-Defined Class Files.....	1393
	– User-Defined Jar Files.....	1396
	– Java: Constructors	1397
	– Java: Static Methods and Static Fields	1397
	– Java: Instance Methods and Instance Fields.....	1398
	– Datatypes: XPath/XQuery to Java.....	1398
	– Datatypes: Java to XPath/XQuery.....	1399
1.5.2	.NET Extension Functions.....	1400
	– .NET: Constructors.....	1402
	– .NET: Static Methods and Static Fields.....	1402
	– .NET: Instance Methods and Instance Fields.....	1403
	– Datatypes: XPath/XQuery to .NET	1404
	– Datatypes: .NET to XPath/XQuery.....	1405
1.5.3	MSXSL Scripts for XSLT.....	1405
1.5.4	Altova Extension Functions.....	1407
	– General Functions.....	1408
	– Chart Functions.....	1411
	– Chart Data XML Structure.....	1414
	– Example: Chart Functions	1419
2	Datatypes in DB-Generated XML Schemas	1423
2.1	MS Access	1424
2.2	MS SQL Server	1425
2.3	MySQL	1426
2.4	Oracle	1427
2.5	ODBC	1428
2.6	ADO	1429
2.7	Sybase	1430
3	Datatypes in DBs Generated from XML Schemas	1431
3.1	MS Access	1432
3.2	MS SQL Server	1434
3.3	MySQL	1436
3.4	Oracle	1438

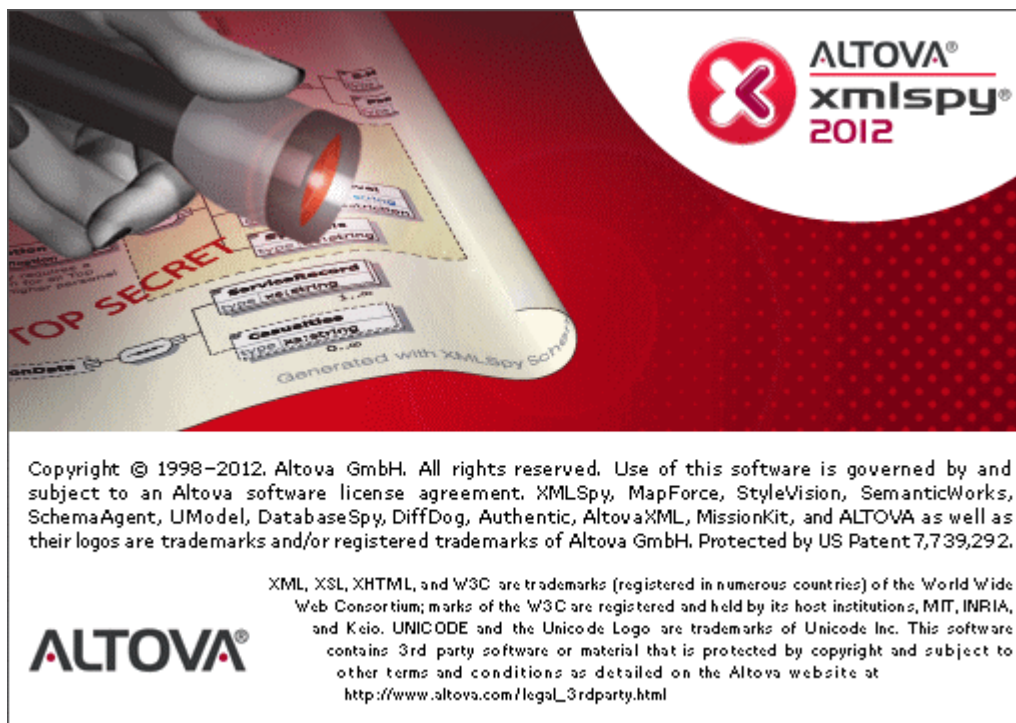
4	Technical Data	1440
4.1	OS and Memory Requirements	1441
4.2	Altova XML Parser	1442
4.3	Altova XSLT and XQuery Engines	1443
4.4	Unicode Support	1444
4.4.1	Windows XP	1444
4.4.2	Right-to-Left Writing Systems	1445
4.5	Internet Usage	1446
5	License Information	1447
5.1	Electronic Software Distribution	1448
5.2	Software Activation and License Metering	1449
5.3	Intellectual Property Rights	1450
5.4	Altova End User License Agreement	1451
	Index	1463

Altova XMLSpy 2012

Welcome to XMLSpy

Welcome to XMLSpy

Altova XMLSpy® 2012 Enterprise Edition is the industry standard XML Development Environment for designing, editing and debugging enterprise-class applications involving XML, XML Schema, XSLT, XQuery, SOAP, WSDL and Web service technologies. It is the ultimate productivity enhancer for J2EE, .NET and database developers. XMLSpy is available in 64-bit and 32-bit versions.



This documentation is organized into the following sections:

- [User Manual](#)
- [Programmers' Reference](#)
- [Appendices](#)

It is available in the following formats:

- As the built-in Help system of XMLSpy ([Help menu](#) or **F1**)
- In HTML and PDF formats, and for purchase as a book, these formats being available via the [Altova website](#)

Last updated: 02/17/2012

Altova XMLSpy 2012

User Manual

User Manual

The User Manual part of this documentation contains all the information you need to get started using [XMLSpy](#) and to learn the various [XMLSpy](#) features. It is organized into four broad parts: (i) an [Introduction](#); (ii) a [Tutorial](#); (iii) a Working With part; and (iv) a [User Reference](#).

We suggest that you start by reading the Introduction in order to get a feel for the GUI and to understand key application settings. If you are new to XML, the [XMLSpy tutorial](#) will help you not only to get to know XMLSpy but also to easily create and use your first XML documents. After that, you should read the [Editing Views section](#) and then the sections in the [Working With part](#) that are of most interest to you. The [User Reference](#) part can be used thereafter as a reference.

Introduction

The [introduction](#) describes the GUI, important settings in the Options dialog, and the application environment.

Tutorial

The [XMLSpy tutorial](#) helps you get started and shows you how to use the most common XMLSpy features.

Working With

The Working With part of the documentation describes the functionality that XMLSpy offers for working with various XML and XML-related technologies. It starts with a description of XMLSpy's [multiple editing views](#). The sections that immediately follow explain the functionality for each technology separately. The Working With part concludes with a series of descriptions of application-wide XMLSpy features, such as Altova Global Resources and projects. The sections in the Working With part are:

- [Editing Views](#): Describes the various editing views in XMLSpy
- [XML](#): Explains the various features available for working with XML documents in XMLSpy
- [DTDs and XML Schemas](#): Shows how schemas (DTDs and XML Schemas) can be edited and leveraged in XMLSpy
- [XSLT and XQuery](#): Presents the range of features available for XSLT and XQuery development
- [Authentic](#): Explains the utility of Altova's graphical XML editor, and shows how it is used
- [HTML, CSS, and JSON](#): Explores XMLSpy's support for HTML, CSS, and JSON
- [WSDL and SOAP](#): Describes XMLSpy's mechanisms for working with these protocols
- [XBRL](#): Describes how to work with the Taxonomy Editor in the XBRL View of XMLSpy
- [Office Open XML and ZIP files](#): Presents and explains Archive View, in which documents using these standards can be edited and managed
- [Databases](#): Explores the wide range of features XMLSpy offers for interfacing XML with databases
- [Global Resources](#): Describes a unique Altova mechanism that can be used to boost development efficiency when using Altova products, especially as a suite of products
- [Projects](#): Explains XMLSpy's project mechanism, which enables increased efficiency and additional development options
- [File/Directory Comparisons](#): Presents XMLSpy's inbuilt comparison engine
- [XMLSpy in Visual Studio](#): Describes integration in the Visual Studio developer platform
- [XMLSpy in Eclipse](#): Describes integration in the Eclipse developer platform

- [Code Generator](#): Explains the working of the built-in code-generator, which produces program code for schema definitions

User Reference

The [User Reference](#) part is organized according to the menus in XMLSpy and describes each menu command in detail.

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 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

1 New Features

Features that are new in XMLSpy **Version 2012** are listed below. Features that were new in older versions are listed further below.

Version 2012 Release 2

- An EPUB file is a zipped group of files used for the distribution of digital publications (EPUB books). In [Archive View](#), you can open EPUB files, create and edit EPUB files, preview the digital EPUB book, edit component files of the EPUB archive directly in XMLSpy, validate the EPUB file, and save the component files back to the EPUB archive. See the section [EPUB Files](#) for details.
- XML special characters can be [escaped or unescaped](#) using a context menu command.
- Bug fixes.

Version 2012

- [Find in Projects](#) enables project files and folder to be quickly found in the Project window.
- Entry helpers and intelligent editing [support for CSS 3.0](#) has been added. This is in addition to support for CSS 2.1.
- Support for [HTML 5.0](#).
- Support for [JDBC connections to databases](#).
- New [Application API interface for Java](#).

Version 2011 Release 3

- Simplified [spellchecker](#).
- In the Enterprise edition of XMLSpy, XML files can be [digitally signed](#) from within the interface. Signatures can be placed instantly within the XML file or can be saved separately. Signatures can also be verified in XMLSpy.
- [Scripting and the Scripting Editor](#) in XMLSpy support for .NET version 4.0 (in addition to versions 2.0, 3.0, and 3.5).

Version 2011 Release 2

- In Schema View, [documentation of the XML Schema document can be generated using an SPS](#) that can be customized in Altova's StyleVision product. In earlier versions of XMLSpy, the design of the generated schema documentation was a standard default design. With this new feature, the structure and formatting of the documentation can be designed by the user.
- The content of an external file, such as an image file, can be encoded as Base-16 or Base-64 text and inserted in Text View and Grid View. This is done via the [Insert | Encoded External File](#) command of the Edit menu.
- Schema View has been enhanced with the following features: (i) [Sorting of global components, attributes, and identity constraints](#); sorting has also been introduced in [Content Model View](#); (ii) [Finding and renaming of global components](#) in the schema file and in related schema files; (iii) If the [base type of a derived type is modified](#), the content, attributes, facets and sample values defined within the derived type can be

- preserved if they are still applicable with the new base type.
- In WSDL View, [documentation of the WSDL document can be generated using an SPS](#) that can be customized in Altova's StyleVision product. In earlier versions of XMLSpy, the design of the generated WSDL documentation was a standard default design. With this new feature, the structure and formatting of the documentation can be designed by the user.
- The [Charts](#) feature has been extended to include stacked bar charts, area charts, stacked area charts, candlestick charts and overlay charts. Charts can be generated for an XML document in Text View and Grid View.
- In XBRL View, [documentation of the XBRL document can be generated using an SPS](#) that can be customized in Altova's StyleVision product. In earlier versions of XMLSpy, the design of the generated XBRL documentation was a standard default design. With this new feature, the structure and formatting of the documentation can be designed by the user.

Version 2011

- In Schema View, creating [Schema Subsets](#) from the components of a single schema file. A schema file can therefore easily be split up into subsets. The reverse process of assimilating components from included schema subsets into the active file is also supported.
- [Charts](#) (such as pie charts and bar charts) that represent data in an XML document can be generated for documents viewed in Text View and Grid View (*Enterprise edition only*). Also, [XSLT/XQuery Profiler results](#) can also be reported in the form of charts (*Enterprise edition only*).
- [Validation of SOAP messages against WSDL](#). SOAP messages can be checked for validity not only against the SOAP specification but as well as against any XML Schemas referenced in the corresponding WSDL definition (*Enterprise edition only*).

Version 2010 Release 3

- Creating [Schema Rules](#) in Schema View for specifying additional schema constraints. (*Enterprise edition only*.)
- [Generate sample values in sample files](#) that are generated from an XML Schema. (*Enterprise and Professional editions only*.)
- [Integration in Microsoft Visual Studio 2010](#). This extends support to the latest version of Visual Studio, which is in addition to support for versions 2005 and 2008. Support for Visual Studio 2003 has been discontinued. (*Enterprise and Professional editions only*.)

Version 2010 Release 2

- Optimizations for working with large files in Text View and Grid View
- Validation against XML Schemas up to three times faster in Text View
- [SharePoint® server support](#) enables files to be checked out and checked in directly from within the XMLSpy interface.
- [CSS and HTML Info windows](#) provide a quick way, via the GUI, to link CSS and HTML files and open them in various browsers and editors.
- [Support for Linux in generated C++ code](#).
- 64-bit versions of XMLSpy available in Enterprise and Professional Editions.

2 Introduction

This introduction describes:

- [The application GUI](#), and
- [The application environment](#).

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

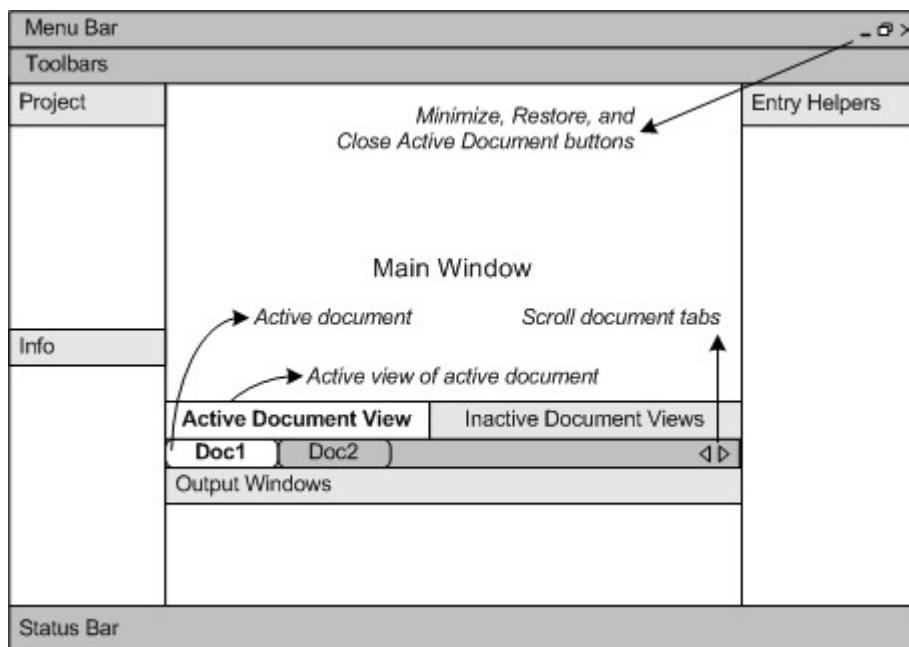
The [Application Environment section](#) points out the various settings that control how files are displayed and can be edited. It also explains how and where you can customize your application. In this section, you will learn where important example and tutorial files have been installed on your machine,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

2.1 The Graphical User Interface (GUI)

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

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

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



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

Switching on and off the display of sidebars

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

Floating and docking the sidebars

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

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

- Right-click the title bar of a window and choose the required command (**Floating** or **Docking**).

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

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

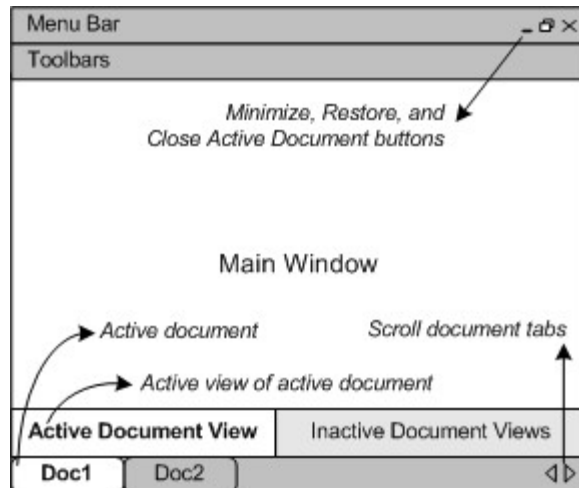
Auto-hiding sidebars

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

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

2.1.1 Main Window

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



Files in the Main Window

- Any number of files can be opened and edited at once.
- Each open document has its own window and a tab with its name at the bottom of the Main Window. To make an open document active, click its tab.
- If several files are open, some document tabs might not be visible for lack of space in the document tabs bar. Document tabs can be brought into view by: (i) using the scroll buttons at the right of the document tab bar, or (ii) selecting the required document from the list at the bottom of the [Window](#) menu.
- When the active document is maximized, its **Minimize**, **Restore**, and **Close** buttons are located at the right side of the Menu Bar. When a document is cascaded, tiled, or

minimized, the **Maximize**, **Restore**, and **Close** buttons are located in the title bar of the document window.

- When you maximize one file, all open files are maximized.
- Open files can be cascaded or tiled using commands in the [Window](#) menu.
- You can also activate open files in the sequence in which they were opened by using **Ctrl+Tab** or **Ctrl+F6**.
- Right-clicking a document tab opens a context-menu with a selection of **File** commands, such as **Print** and **Close**.

Views in the Main Window

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

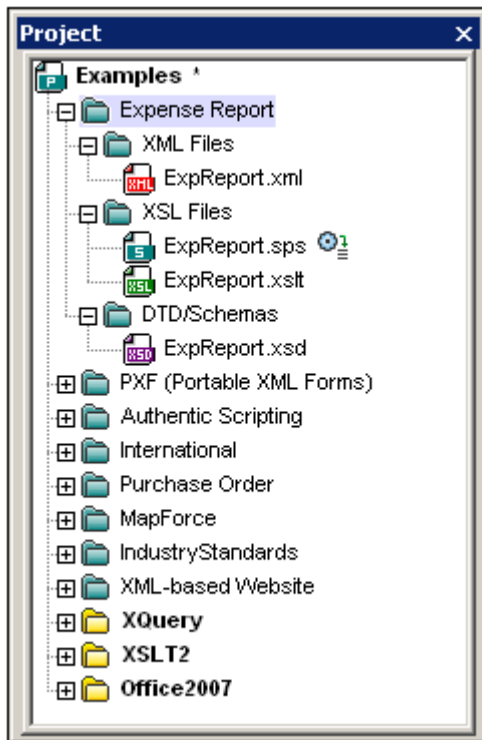
The available views are either editing or browser views:

- [Text View](#): An editing view with syntax-coloring for source-level work.
- [Grid View](#): For structured editing. The document is displayed as a structured grid, which can be manipulated graphically. This view also contains an embedded [Table view](#), which shows repeating elements in a tabular format.
- [Schema View](#): For viewing and editing XML Schemas.
- [WSDL View](#): For viewing and editing WSDL documents.
- [Authentic View](#): For editing XML documents that are based on StyleVision Power Stylesheets
- [Browser View](#): An integrated browser view that supports both CSS and XSL stylesheets.

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

2.1.2 Project Window

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



Projects thus enable you to gather together files that are used together and to access them quicker. Additionally, you can define schemas and XSLT files for individual folders, thus enabling the batch processing of files in a folder.

Project operations

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

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

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

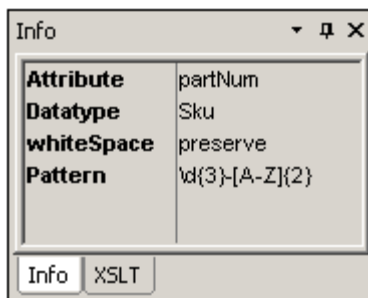
For a more detailed description of projects, see the section [Projects](#).

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

2.1.3 Info Window

The Info Window (*screenshot below*) shows information about the element or attribute in which the cursor is currently positioned.

- When an XSLT document is active, additional XSLT-specific information and commands are available in the XSLT tab of the Info window (*see XSLT tab in screenshot below*). How to read the information and use the commands in the XSLT tab is explained in the section [XSLT and XQ | XSLT | XSL Outline](#).
- When an XML Schema document (.xsd file) is active, options for enabling extended validation are available in the Schema tab of the Info window. How to set up extended validation is described in the section, [Schema Rules](#).



Information is available in the Info Window in Text View, Grid View, and Authentic View.

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

2.1.4 Entry Helpers

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

The entry helpers that are available depend upon:

1. *The kind of document being edited.* For example, XML documents will have different entry helpers than XQuery documents: elements, attributes, and entities entry helpers in the former case, but XQuery keywords, variables, and functions entry helpers in the latter case. The available entry helpers for each document type are described in the description of that document type.
2. *The current view.* Since the editing mechanisms in the different views are different, the entry helpers are designed so as to be compatible with the editing mechanism in the

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

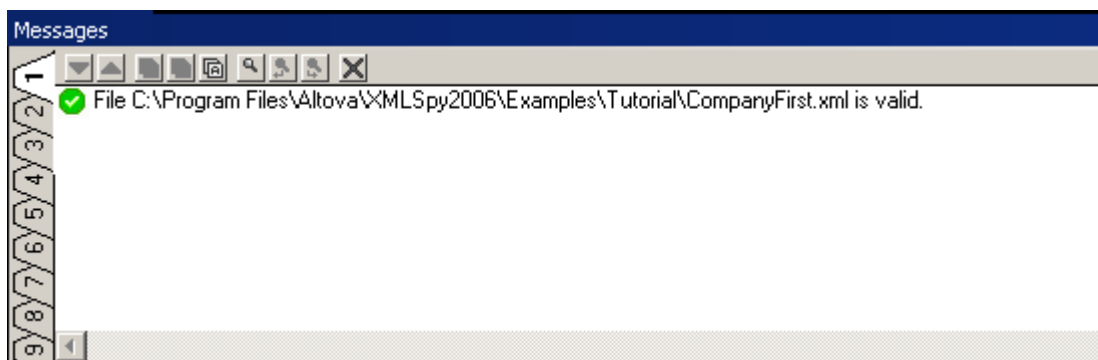
A general description of entry helpers in each type of view is given in [Editing Views](#). Further document-type-related differences within a view are noted in the description of the individual document types, for example [XML entry helpers](#) and [XQuery entry helpers](#).

Note the following:

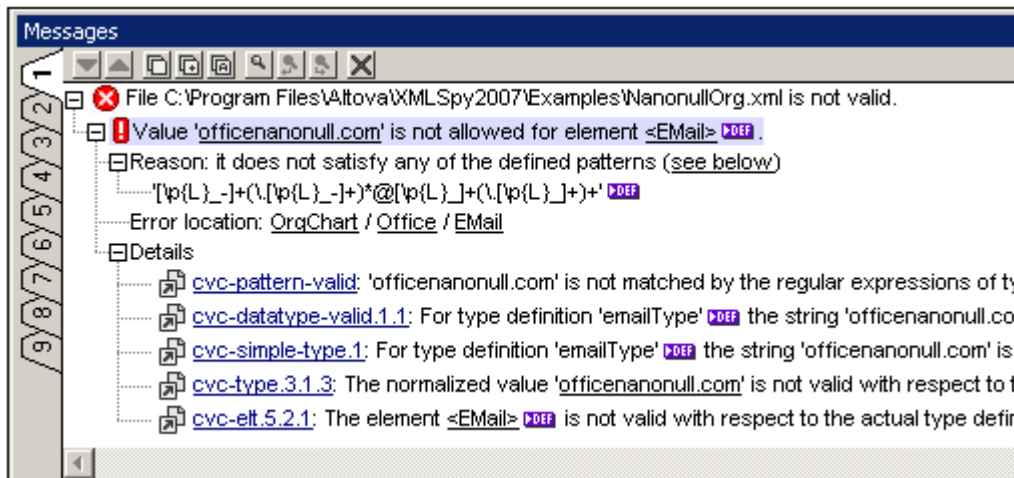
- You can turn the display of entry helpers on or off with the menu option **Window | Entry Helpers**.
- In Visual Studio .NET, entry helper windows have a prefix that is the application name.

2.1.5 Output Window: Messages

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



Otherwise, a message that describes the error (*screenshot below*) is displayed. Notice that there are links (black link text) to nodes and node content in the XML document, as well as links (blue link text) to the sections in the relevant specification on the Internet that describe the rule in question. Clicking the purple `Def` buttons, opens the relevant schema definition in Schema View.

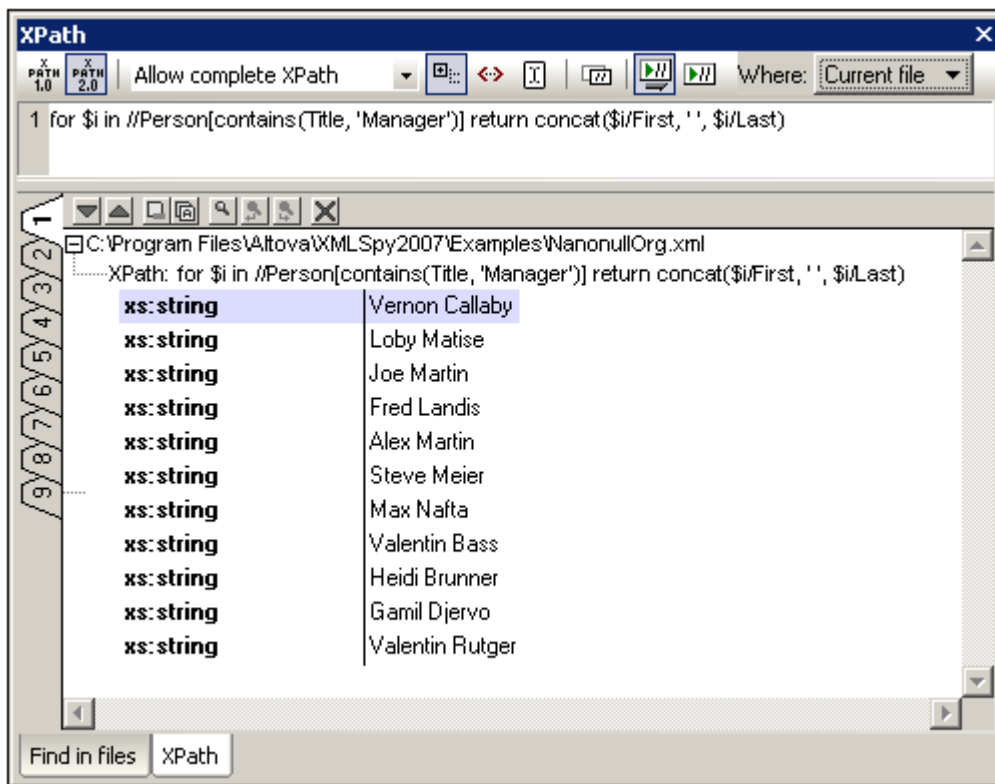


The Messages Window is enabled in all views, but clicking a link to content in an XML document highlights that node in the XML document in Text View. However, when an XML Schema has been validated in Schema View, clicking a `Def` button does not change the view.

Note: The **Validate** command (in the XML menu) is normally applied to the active document. But you can also apply the command to a file, folder, or group of files in the active project. Select the required file or folder in the Project Window (by clicking on it), and click **XML | Validate** or **F8**. Invalid files in a project will be opened and made active in the Main Window, and the File Is Invalid error message will be displayed.

2.1.6 Output Window: XPath

The XPath Window enables you to evaluate up to nine XPath expressions, each in the context of the currently active XML document. Additionally, an expression can be evaluated in: (i) all currently open XML documents; (ii) XML files of the currently active XMLSpy project; and (iii) XML files of a selected folder. This means that you can enter different XPath expressions in different tabs and then evaluate each expression on multiple XML files at once.

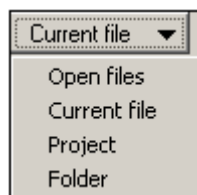


The XPath Window shows, in a single view: (i) the XML document; (ii) the XPath expression; and (iii) the result of evaluating the XPath expression on the active XML document. The benefits are that you can: (i) navigate the XML document while keeping the XPath expression and its results in view; (ii) navigate the XML document by clicking items in the result; and (iii) modify the XPath expression while keeping the XML document in view.

Steps for evaluating an XPath expression

Given below are the main steps required to use the XPath Window.

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

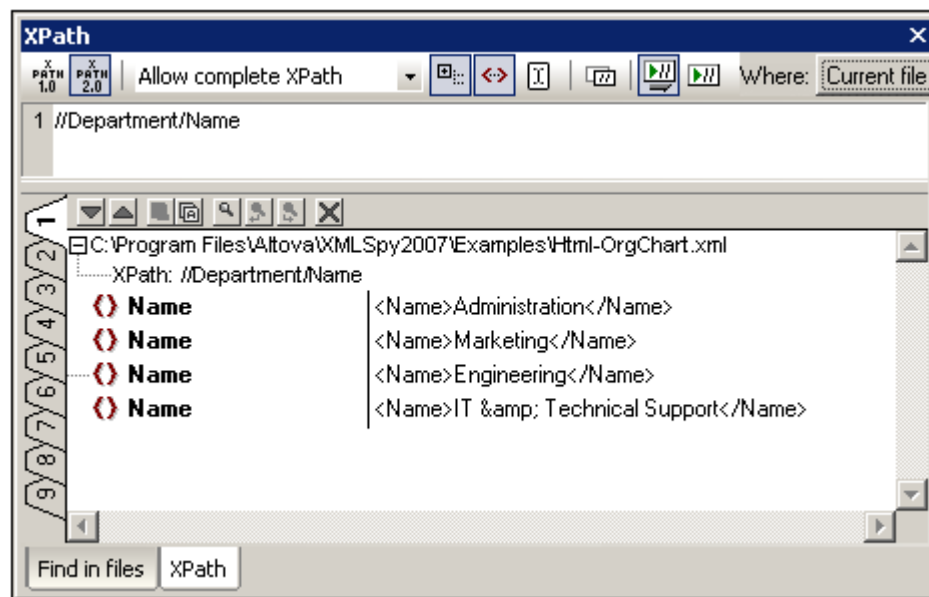


If *Current file* is selected, the file that is currently active is used. Selecting *Open files* causes the XPath expression to be evaluate against all the files currently open in XMLSpy. Project refers to the currently active XMLSpy project. The external folders in a XMLSpy project can be excluded by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

2. Select the XPath version you wish to use (1.0 or 2.0) by clicking the appropriate icon in the toolbar of the output window (*see screenshot below*).
3. Select the type of XPath expression from the dropdown list in the combo box. *Allow*

Complete XPath is the usually required option. The XML Schema Selector and XML Schema Field options can be used for a narrow subset of specific XPath 1.0 cases and are useful when unique identity constraints have been defined in the XML Schema. When either of these options is selected, only name tests (and the wildcard *) are allowed in the XPath expression, and predicates and XPath functions may not be used. Furthermore, for the XML Schema Selector option, only expressions on the child axis are allowed; for the XML Schema Field option, expressions on the child axis and attribute axis are allowed. For more information, see the W3C's [XML Schema: Structures Recommendation](#).

4. Toggle the Evaluate XPath Expression On Typing icon  on if you want the XPath expression to be evaluated while you are typing it in. If this icon is toggled off, the expression will be evaluated only when you click the Evaluate XPath Expression icon .
5. Toggle the Show Header In Output icon  on if, in the output, you wish to show the location of the XML file and the XPath expression (as in screenshot below).
6. If the XPath expression will returns nodes—such as elements or attributes—you can select whether the entire contents of the selected nodes should be shown. This is done by switching the Show Complete Results icon  on. In the screenshot below, both the element and its content are displayed (in the right-hand column).



7. To set an XPath expression relative to a selection in the XML document, toggle the Set Current Selection As Origin icon  on.
8. Enter the XPath expression. The intelligent XPath editing feature will pop up a list of XPath functions, XPath axes, and document elements and attributes, from which you can choose. If you wish to create the expression over multiple lines, press the **Return** key.
9. To evaluate the expression (if Evaluate on Typing is toggled off or when a new XML document is made active), click the Evaluate XPath Expression icon.

Features of the Result Pane

The Result Pane has the following useful features:

- When the result contains a node (including a text node)—as opposed to expression-generated literals—clicking on that node in the Result Pane highlights the

- corresponding node in the XML document in the Main Window.
- When a node in the XML document is selected, clicking the Copies XPath of Current Selection to Edit Field icon  does the following: (i) generates an XPath expression to specifically locate the selected node; (ii) overwrites the expression currently in the edit field with the newly generated XPath expression.
- The Copies XPath from Current Result Tab to Edit Field icon  copies the XPath expression of the currently open result tab to the XPath expression edit field. This is useful if (i) the expression in the edit field is different than one that was used to generate the result in one of the result tabs, and (ii) you wish to re-evaluate the expression used in that result tab. In this case, you make the result tab active and then use this command to copy the result tab's XPath expression to the edit field. The XPath expression can then be re-evaluated.

XPath expressions in the edit field

The following general points about XPath expressions should be noted. Issues specific to XPath 1.0 and XPath 2.0, respectively, are treated separately below.

- When you enter an expression in the edit field, it is displayed in black if the syntax is correct and in red if the syntax is incorrect. Note that correct syntax does not ensure that the expression is error-free. Error messages are displayed in the results pane.
- An error might be caused because of incorrectly set options. So set the options correctly, especially the XPath Version and the XPath Origin (absolute, or relative to the current selection in the currently active XML document).
- Only namespaces that are in scope on the element where the XPath expression originates (i.e. on the context node) are evaluated.

XPath 1.0 expressions

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

XPath 2.0 expressions

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

`http://www.w3.org/2001/XMLSchema)` must be declared in the XML file and the namespace prefix must be used on these types in the XPath expression. Example:
`years-from-yearMonthDuration( xs: yearMonthDuration( ' P22Y18M' ) )`.

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

Datatypes in XPath 2.0

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

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

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

String length of character and entity references

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

Troubleshooting

If your XPath expression returns an error, do the following:

1. Check whether the options have been correctly set (XPath version, origin, etc).
2. Check that the spelling of function names, constructor names, node names, etc., are correct.
3. Check whether prefixes are required and correctly set on functions, constructors, and arguments in the XPath expression.
4. If namespaces are declared in the XML document check that they are correct. The correct namespaces are:

XML Schema: `http://www.w3.org/2001/XMLSchema`

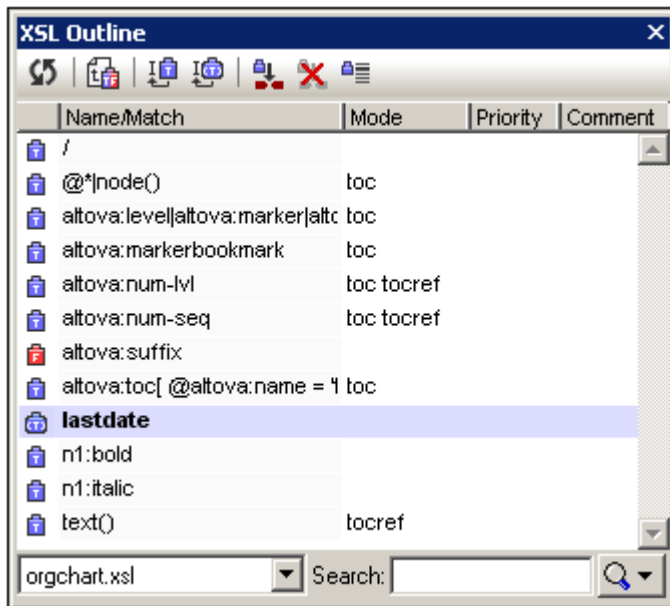
XPath 2.0 Functions: `http://www.w3.org/2005/xpath-functions`

XPath 2.0 Functions Support

See the [appendices](#).

2.1.7 Output Window: XSL Outline

The XSL Outline Window (*screenshot below*) lists all the templates and functions in an XSLT stylesheet, and, optionally, in all included and imported XSLT stylesheets as well. The XSL Outline Window is located by default docked with the Output Windows at the bottom of the XMLSpy window. It can be undocked, or docked along another edge of the XMLSpy window.

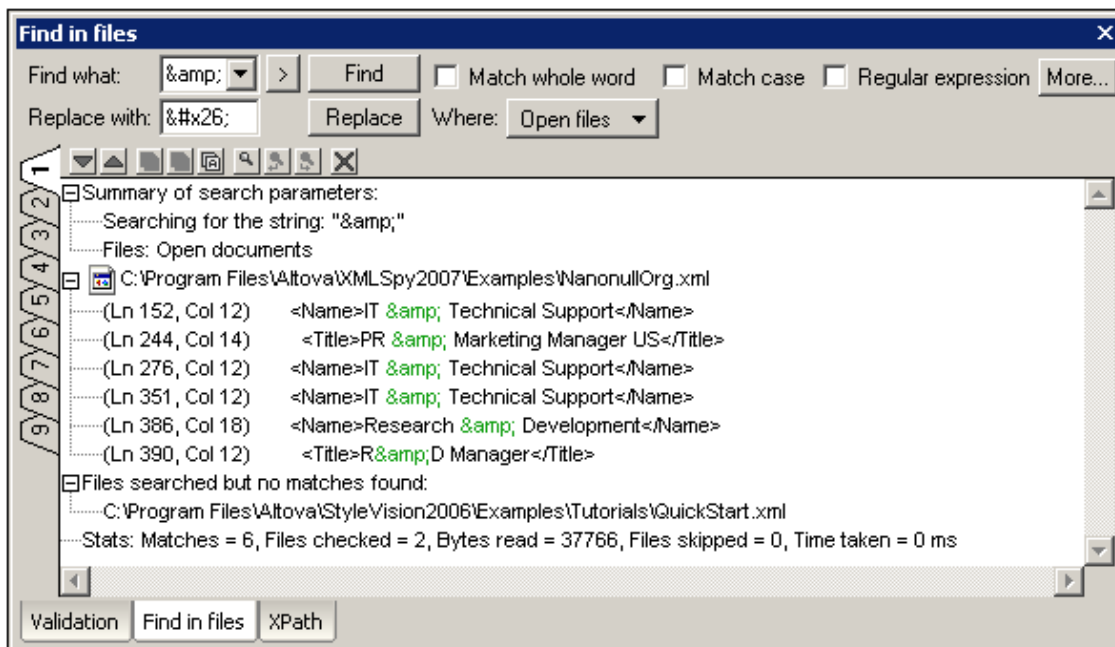


The XSL Outline Window provides information about templates and functions in the stylesheet. This information can be sorted and searched, and the window's toolbar contains commands that enable you to easily insert calls to named templates and to set named templates as the starting point of transformations. How to work with the XSL Outline Window is described in the section [XSLT and XQuery | XSLT | XSL Outline | XSL Outline Window](#).

Note: File-related information about the stylesheet and file-related commands are available in the XSLT tab of the Info Window. How to use these commands is described in the section [XSLT and XQuery | XSLT | XSL Outline | Info Window](#).

2.1.8 Output Window: Find in Files

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



Find criteria

There are two broad find criteria: (i) what to find, and (ii) where to look?

What to find: The string to find is entered in the Find What text box. If that string must match a whole word, then the Match Whole Word check box must be clicked. For example, for the find string `fit`, with Match Whole Word checked, only the word `fit` will match the find string; the `fit` in `fitness`, for example, would not. You can specify whether casing is significant using the Match Case check box. If the text entered in the Find What text box is a regular expression, then the Regular Expression check box must be checked. An entry helper for regular

expressions can be accessed by clicking the  button. The use of regular expressions for searching is explained in the section, [Find](#). The **More** button opens the [Find in Files dialog](#), where you can set advanced search conditions and actions. For more information, see [Edit | Find in Files](#).

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

Replace with

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

The results

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

- A summary of the search parameters, which lists the search string and what files were searched.
- A listing of the found or replaced strings (according to whether the **Find** or **Replace** button was pressed). The items in this listing are links to the found/replaced text in the

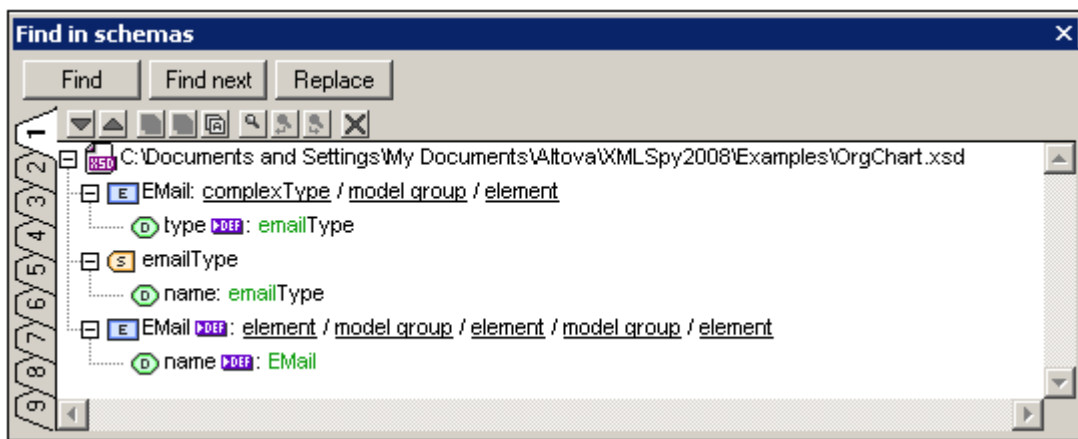
Text View of the document. If the document is not open, it will be opened in Text View and the found/replaced text will be highlighted.

- A list of the files which were searched but in which no matches were found.
- A summary of statistics for the search action, including the number of matches and number of files checked.

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

2.1.9 Output Window: Find in Schemas

When an XML Schema is active in Schema View, it can be searched intelligently using XMLSpy's Find and Replace in Schema View feature. The Find and Replace in Schema View feature is accessed via: (i) the **Find** and **Replace** commands in the **Edit** menu; and (ii) the **Find** and **Replace** buttons in the Find in Schemas Window (*screenshot below*).



The results of the Find and Replace in Schema View feature (i.e. each time a Find or Replace command is executed) are displayed in the Find in Schemas window. The term that was searched for is displayed in green; (in the screenshot above, it can be seen that `email` was the search term, with no case restriction specified). Notice that the location of the schema file is also given.

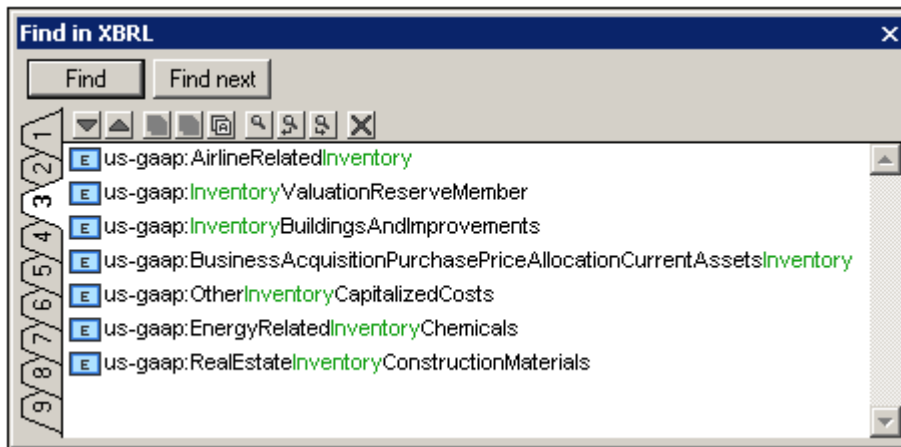
Results are displayed in nine separate tabs (numbered 1 to 9). So you can keep the results of one search in one tab, do a new search in a new tab, and compare results. To show the results of a new search in a new tab, select the new tab before starting the search. Clicking on a result in the Find In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

2.1.10 Output Window: Find in XBRL

The Find in XBRL Window (*screenshot below*) displays the results of searching an XBRL taxonomy document. There are nine tabs in this window, so results in one tab can be compared with the results in another tab.

The Find in XBRL can be performed when an XBRL taxonomy document is open in XBRL View. How to carry out the search is described in the section [Find in XBRL](#) in the [XBRL section](#) of the

user manual.



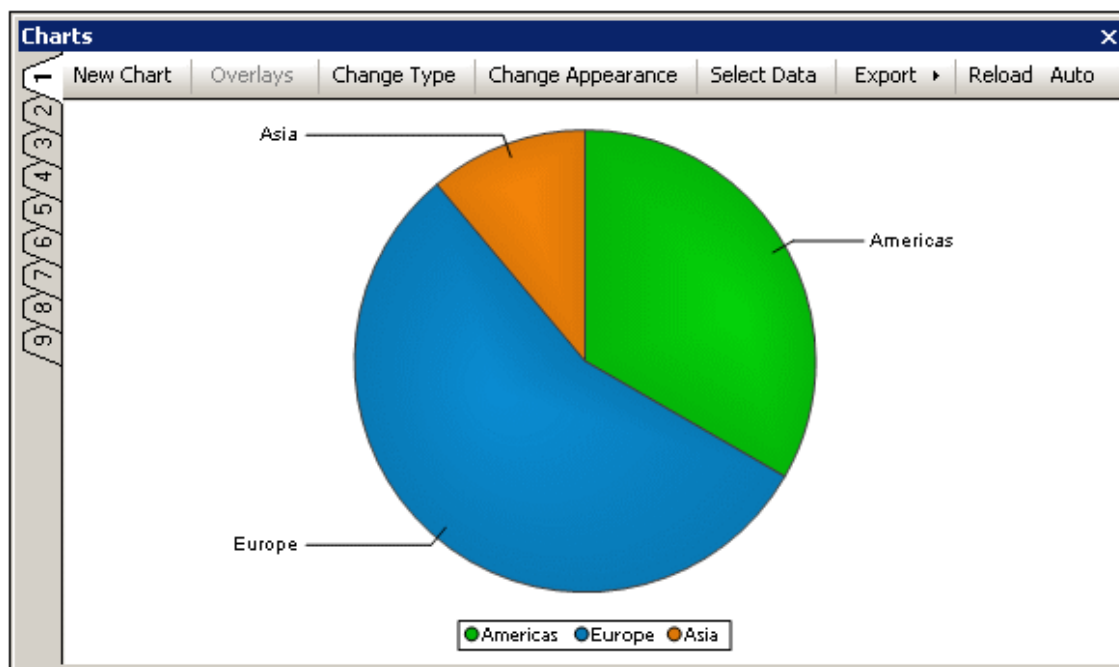
The **Find** button pops up the Search dialog. The **Find Next** button finds the next instance of the search term in the document starting from the cell immediately after the cell in XBRL View in which the cursor is currently placed.

The following Find In XBRL toolbar commands are available:

- The **Next** and **Previous** icons select, respectively, the next and previous find results to the currently selected result.
- The **Copy Messages** commands copy, respectively, the selected message, the selected message and its children messages, and all messages, to the clipboard.
- The **Find** commands find text strings in the Find In XBRL window.
- The **Clear** command deletes all messages in the currently active tab.

2.1.11 Output Window: Charts

When an XML document is open in Text View or Grid View, a chart (pie chart, bar chart, etc) representing selected data in the XML document can be generated in the Charts Window (*screenshot below*).



Creating the chart

The broad steps for creating a chart are as follows:

1. Place the cursor in the XML document to select a context node.
2. Click the **New Chart** button in the Charts Window (see screenshot above) or right-click in the Main Window and select **New Chart** from the context menu.
3. In the Select Columns dialog that pops up, select data for the chart data table and click **OK**. The chart will be created in the Charts Window (screenshot above).

For more information, see the section [Charts](#) in the [XML section](#) of the user manual.

Modifying and managing charts

A chart can be created in any of the nine Charts Window tabs (numbered along the left side of the window). In this way charts in different tabs can be compared. A chart created in a tab can only be overwritten when a new chart is created in that tab. A chart cannot be otherwise deleted. Even when the XML document that was used to generate a chart is closed, the chart remains in the tab in which it was created.

The buttons at the top of the window do the following:

- **New Chart:** Pops up the Select Columns dialog, in which the chart data table is configured.
- **Change Type:** Enables the chart type to be changed, for example, from a bar chart to a pie chart.
- **Change Appearance:** Enables settings, like the chart's fonts sizes and color schemes, to be changed.
- **Select Data:** Pops up the Select Data dialog, which contains the chart data table and the final selection of data that will be presented in the chart. Data for the series, the X-Axis and Y-Axis can be modified in this dialog. The X-Axis and Y-Axis data can be graphically selected from the chart data table. Clicking **OK** generates the modified chart in the Charts Window.
- **Export:** the chart can be exported as an image file, or as an XSLT or XQuery fragment

to the clipboard. The XSLT or XQuery fragment can be used in an XSLT or XQuery document, which when processed with the Altova XSLT 2.0 Engine or the Altova XQuery Engine, will correctly render the chart.

- **Reload/Auto:** If the **Auto** button is toggled on, then any change in the underlying XML document will automatically refresh a chart in the Charts Window that is based on the XML document. Otherwise a chart will only be updated when the **Reload** button is pressed.

2.1.12 Menu Bar, Toolbars, Status Bar

Menu Bar

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

- If commands in a menu are **not** applicable in a view or at a particular location in the document, they are unavailable.
- Some menu commands pop up a submenu with a list of additional options. Menu commands with submenus are indicated with a right-pointing arrowhead to the right of the command name.
- Some menu commands pop up a dialog that prompts you for further information required to carry out the selected command. Such commands are indicated with an ellipsis (...) after the name of the command.
- To access a menu command, click the menu name and then the command. If a submenu is indicated for a menu item, the submenu opens when you mouseover the menu item. Click the required sub-menu item.
- A menu can be opened from the keyboard by pressing the appropriate key combination. The key combination for each menu is **Alt+KEY**, where **KEY** is the underlined letter in the menu name. For example, the key combination for the **File** menu is **Alt+F**.
- A menu command (that is, a command in a menu) can be selected by sequentially selecting (i) the menu with its key combination (see previous point), and then (ii) the key combination for the specific command (**Alt+KEY**, where **KEY** is the underlined letter in the command name). For example, to create a new file (**File | New**), press **Alt+F** and then **Alt+N**.
- Some menu commands can be selected **directly** by pressing a special **shortcut** key or key combination (**Ctrl+KEY**). Commands which have shortcuts associated with them are indicated with the shortcut key or key combination listed to the right of the command. For example, you can use the shortcut key combination **Ctrl+N** to create a new file; the shortcut key **F8** to validate an XML file. You can [create your own shortcuts](#) in the Keyboard tab of the Customize dialog (**Tools | Customize**).

Toolbars

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

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

Status Bar

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

2.2 The Application Environment

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

This section is organized as follows:

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

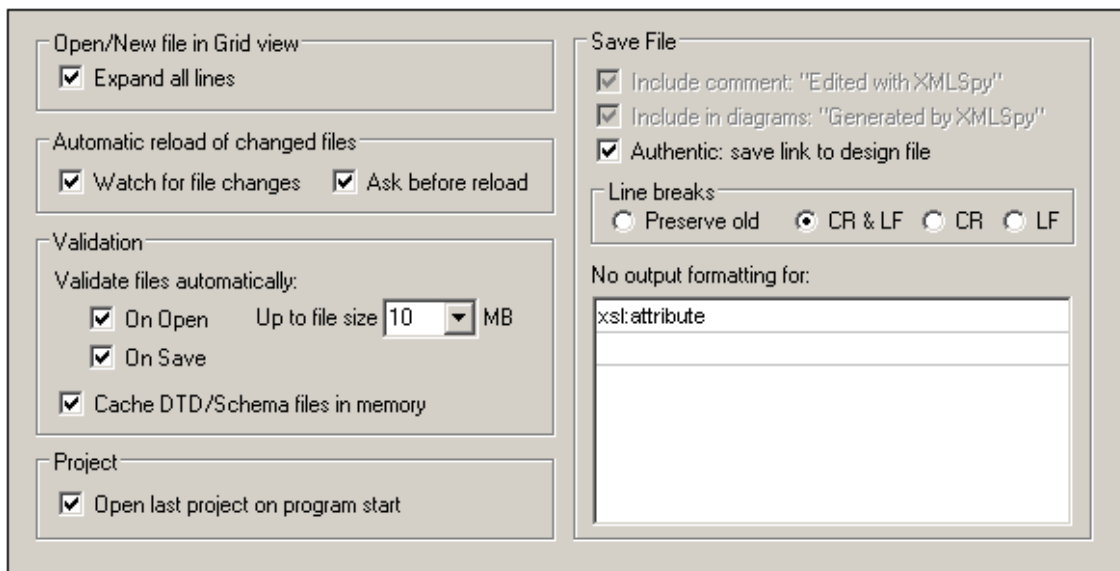
2.2.1 Settings, Customization, and Menus

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

- [Settings](#)
- [Customization](#)
- [Menus](#)

Settings

Several important XMLSpy settings are defined in different tabs in the Options dialog (*screenshot below*, accessed via the menu command [Tools | Options](#)). You should look through the various options to familiarize yourself with what's available.

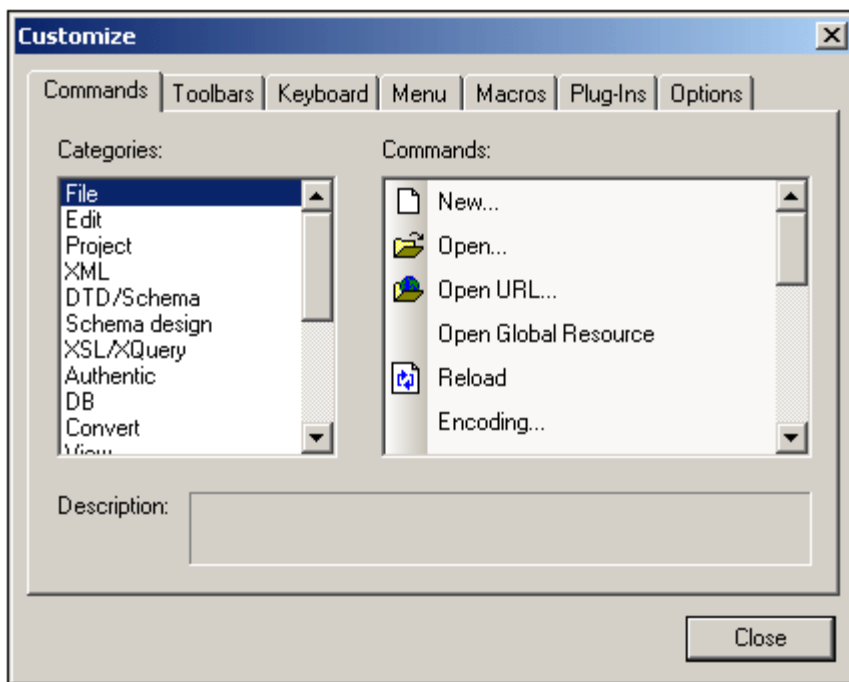


Given below is a summary of the most important settings. For details, see the description of the [Options dialog](#) in the User Reference section.

- *File types and default views:* In the File Types tab, you can add a file type that XMLSpy will recognize. You do this by adding a file extension to the list of file extensions. For each file type, you can then specify to what specification files of this file type should conform and in what XMLSpy view files of this file type should open (the default view for this file type).
- *File validation:* In the File tab, you can specify whether files should be validated automatically on opening and/or saving. In the File Types tab, file validation can then be disabled for specific file types.
- *Editing features:* In the Editing tab, you can specify how entry helpers should be organized, how new elements are generated, and whether auto-completion is enabled. Additional options are available for individual views in the View tab. In the Fonts tabs for various views, you can specify the font characteristics of individual node types in each of these views.
- *XSLT and FO Engines:* In the XSL tab, you can specify that an external XSL engine be used for transformations made from within the GUI. You must also specify the location of the FO processor executable to be used for FO processing within XMLSpy. For more information, see the [XSLT Processing](#) section.
- *Encoding:* Default encodings for XML and non-XML files are specified in the Encoding tab.

Customization

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



The various customization options are described in the [User Reference](#) section.

Menus

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

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

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

2.2.2 Tutorials, Projects, Examples

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

Location of tutorials, projects, and example files

The XMLSpy tutorials, projects, and example files are installed in the folder:

```
C:\Documents and Settings\<username>\My Documents\  
Altova\XMLSpy2012\Examples\
```

The `My Documents\Altova\XMLSpy2012` folder will be installed for each user registered on a PC within that user's `<username>` folder. Under this installation system, therefore, each user will have his or her own `ExamplesAuthenticExamples` folder in a separate working area.

Note about the master XMLSpy folder

When XMLSpy is installed on a machine, a master `Altova\XMLSpy2012` folder is created at the following folder location:

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

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

Location of tutorial, project, and examples files

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

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

2.2.3 XMLSpy Features and Help, and Altova Products

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

XMLSpy feature listing

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

XMLSpy Help

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

- [Online HTML manuals](#), accessed via the Support page at the Altova website
- [Printable PDFs](#), which you can download from the Altova website and print locally
- [Printed books](#) that you can buy via a link at the Altova website

Support options

If you require additional information to what is available in the user manual (this documentation) or have a query about Altova products, visit our [Support Center](#) at the Altova website. Here you will find:

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

Altova products

For a list of all Altova products, see the [Altova website](#).

3 XMLSpy Tutorial

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

The tutorial is divided into the following parts:

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

Installation and configuration

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

Tutorial example files

The tutorial files are available in the application folder:

```
C:\Documents and Settings\<username>\My Documents\Altova\XMLSpy2012\
Examples\Tutorial
```

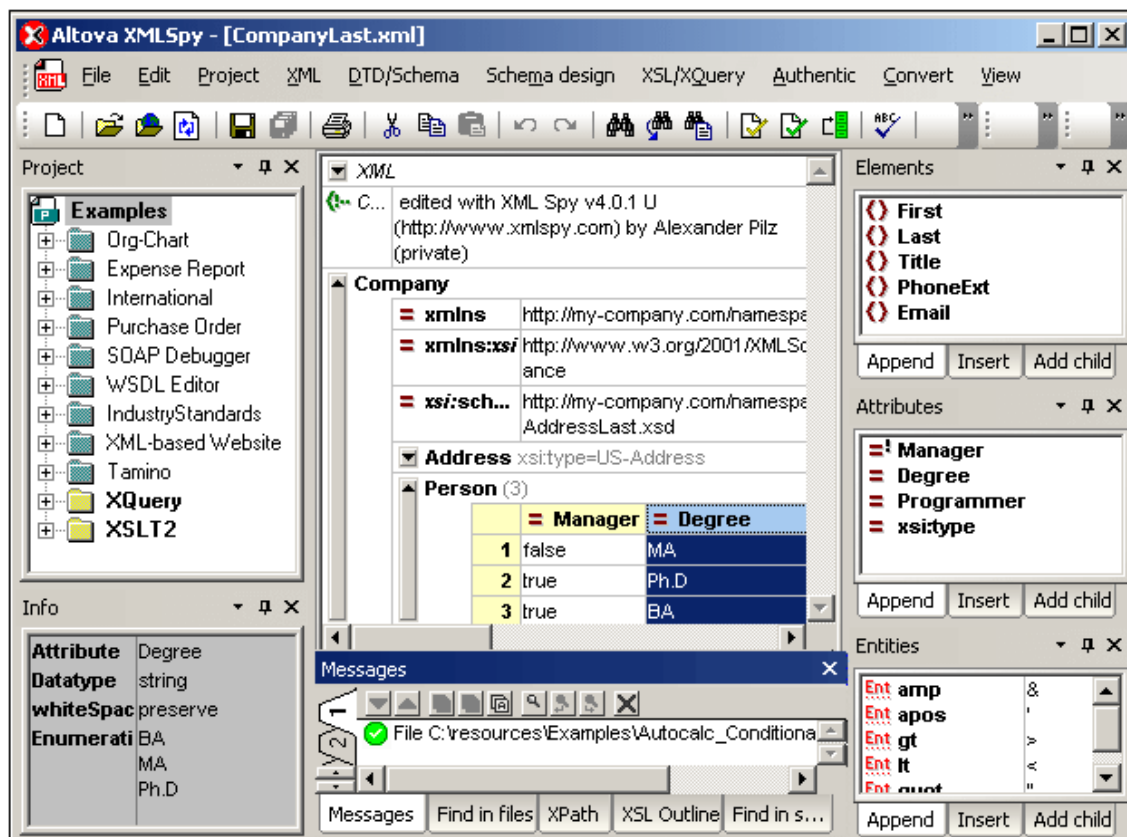
The `Examples` folder contains various XML files for you to experiment with, while the `Tutorial` folder contains all the files used in this tutorial.

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

3.1 XMLSpy Interface

The XMLSpy interface is structured into three vertical areas. The central area provides you with multiple views of your XML document. The areas on either side of this central area contain windows that provide information, editing help, and file management features.

- The left area consists of the **Project** and **Info** windows.
- The central area, called the **Main** window, is where you edit and view all types of XML documents. You can switch between different views: [Text View](#), [Grid View](#), [Schema View](#), [WSDL View](#), [Authentic View](#), and [Browser View](#). These views are described in detail in the individual sections about them in the User Manual.
- The right-hand area contains the three **Entry Helper** windows, which enable you to insert or append elements, attributes, and entities. What entries are displayed in the Entry Helper windows depends on the current selection or cursor location in the XML file.



The details of the interface are explained as we go along. Note that the interface changes dynamically according to the document that is active in the Main Window and according to the view selected.

3.2 XML Schemas: Basics

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

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

Objective

In this section of the tutorial, you will learn how to edit XML Schemas in Schema View. Specifically, you will learn how to do the following:

- Create a new schema file
- Define namespaces for the schema
- Define a basic content model
- Add elements to the content model using context menus and drag-and-drop
- Configure the Content Model View

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

Commands used in this section

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



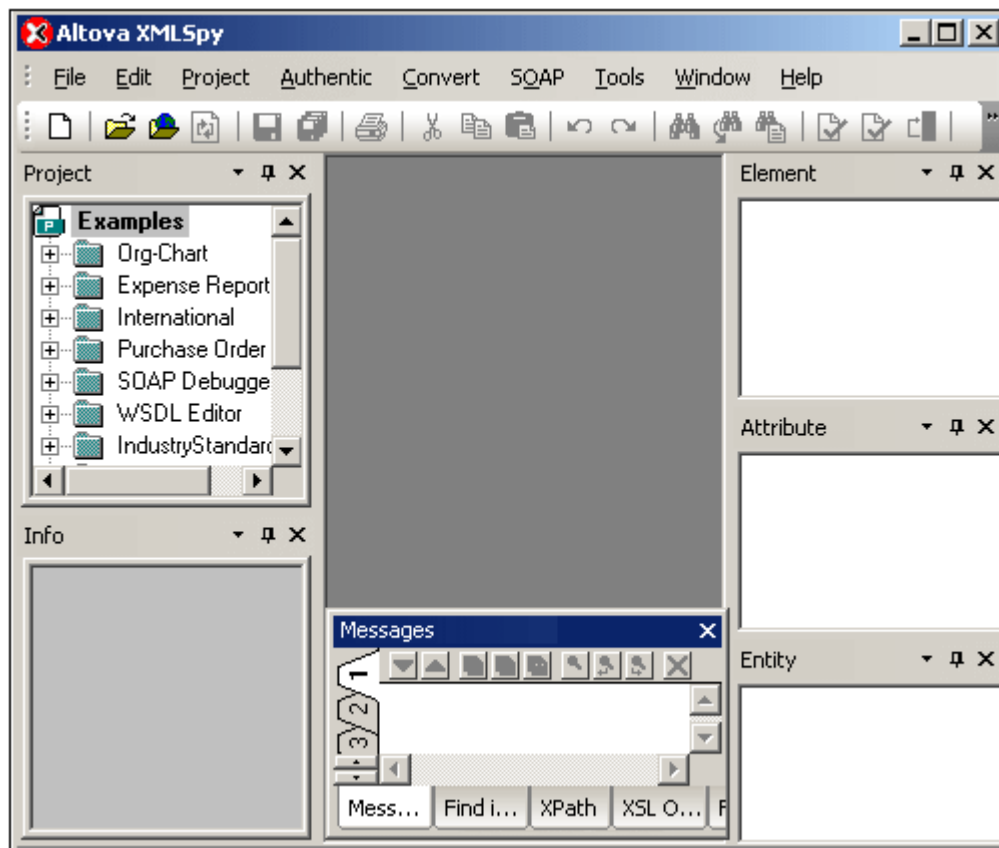
Display Diagram (or Display Content Model View). This icon is located to the left of all global components in Schema Overview. Click this icon to display the content model of the associated global component.

3.2.1 Creating a New XML Schema File

To create a new XML Schema file in XMLSpy, you must first start XMLSpy and then create a new XML Schema (.xsd) document.

Starting XMLSpy

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

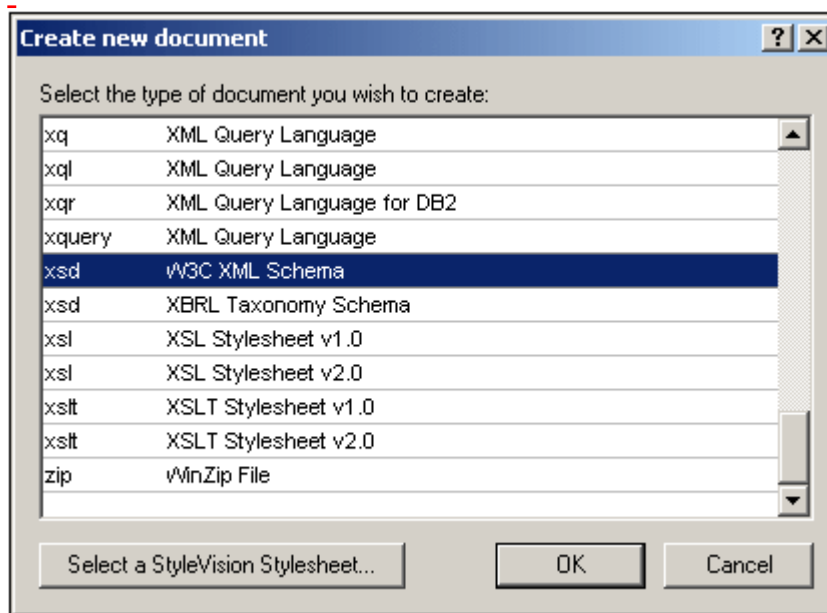


Note the four main parts of the interface: (i) the Project and Info Windows on the left; (ii) the Main Window in the middle; (iii) the Entry Helpers on the right; and (iv) the Output Windows at the bottom.

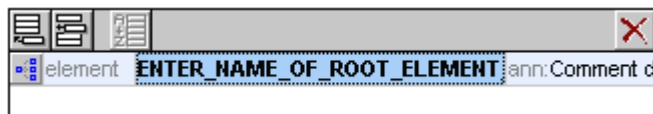
Creating a new XML Schema file

To create a new XML Schema file:

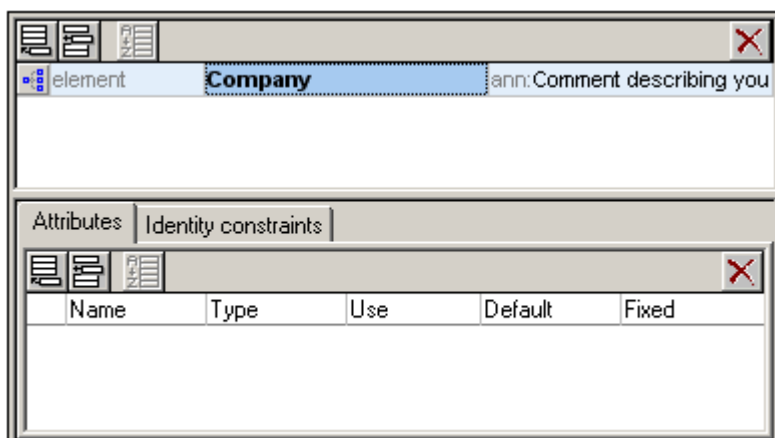
1. Select the menu option **File | New**. The Create new document dialog opens.



2. In the dialog, select the `xsd` entry (the document description and the list in the window might vary from that in the screenshot) and confirm with **OK**. An empty schema file appears in the Main Window in Schema View. You are prompted to enter the name of the root element.



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



4. In the Annotations field (ann) of the `Company` element, enter the description of the element, in this case, `Root element`.
5. Click the menu option **File | Save**, and save your XML Schema with any name you like (`AddressFirst.xsd`, for example).

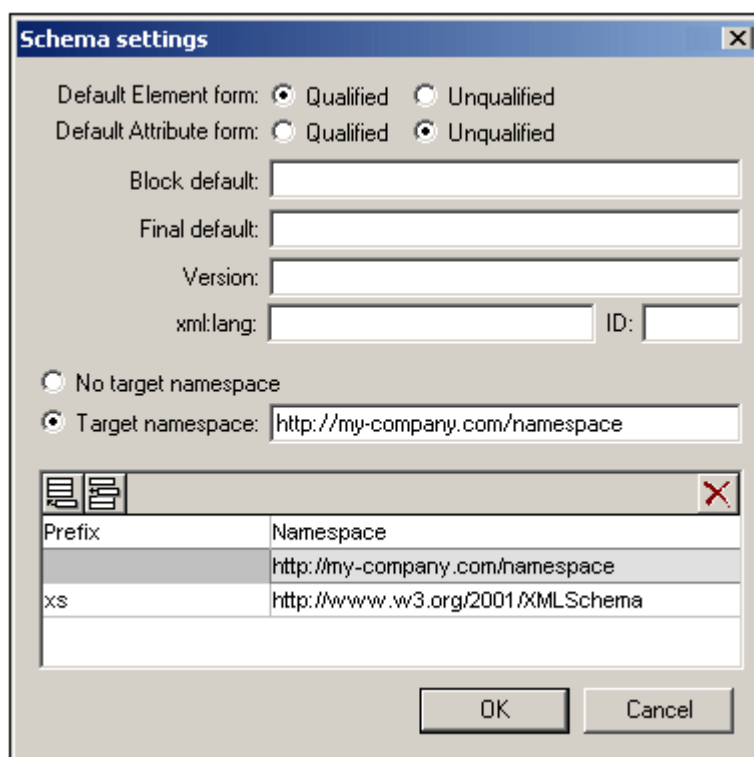
3.2.2 Defining Namespaces

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

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

To create a target namespace:

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



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

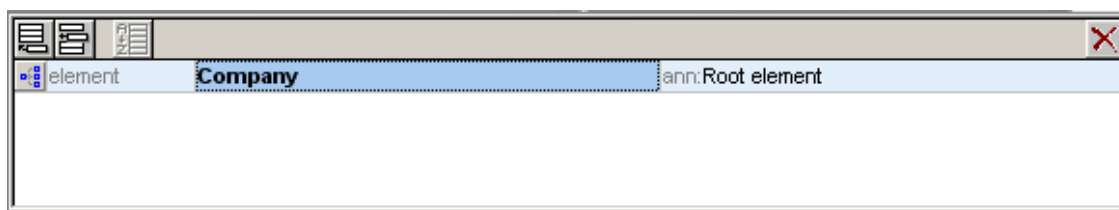
Please note:

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

3.2.3 Defining a Content Model

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

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

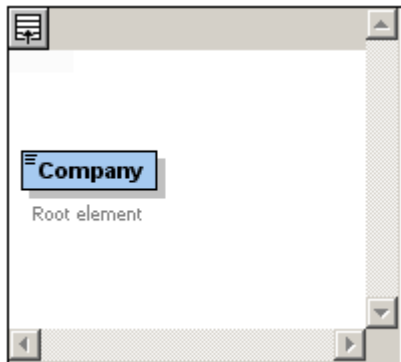


In this section, you will create the content model of the `Company` element.

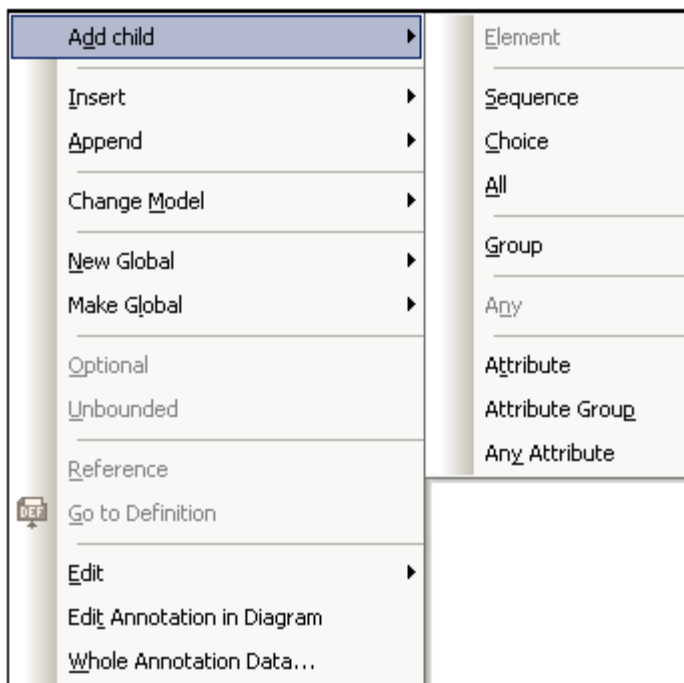
Creating a basic content model

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

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



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



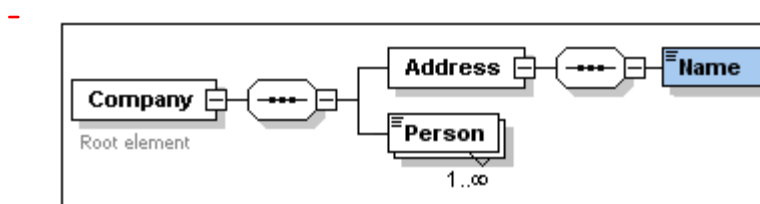
This inserts the Sequence compositor, which defines that the components that follow must appear in the specified sequence.

Adding additional levels to the content model structure

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

Do this as follows:

1. Right-click the `Address` element to open the context menu, and select **Add Child | Sequence**. This adds the Sequence compositor.
2. Right-click the Sequence compositor, and select **Add Child | Element**. Name the newly created element component `Name`.



Complex types, simple types, and XML Schema data types

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

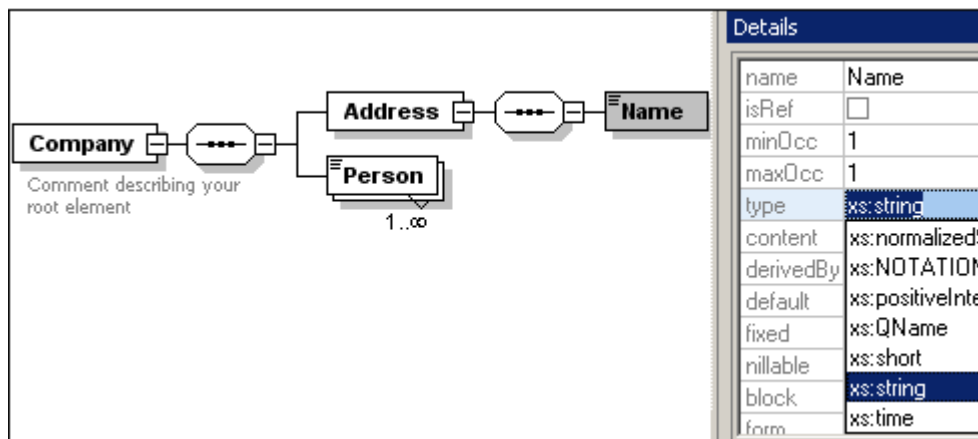
```
<xs:element name="Company">
  <xs:annotation>
    <xs:documentation>Root element</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Address">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Name"/>
          </xs:sequence>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

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

To define the `Name` element to be of this datatype:

1. Click the **Schema** tab to return to Schema View.
2. Click the `Name` element to select it.

3. In the Details Entry Helper, from the dropdown menu of the `type` combo box, select the `xs:string` entry.



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

The text representation of the `Name` element is as follows:

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

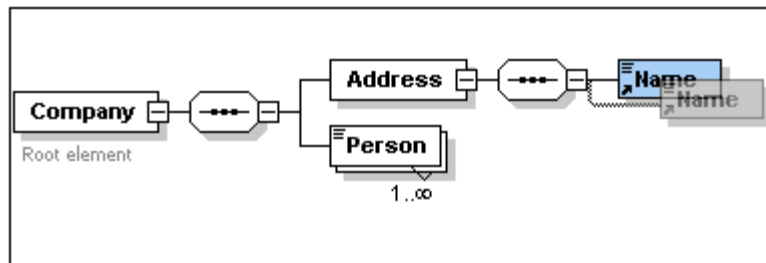
Please note: A simple type element can have any one of several XML Schema data types. In all these cases, the icon indicating text-content appears in the element box.

3.2.4 Adding Elements with Drag-and-Drop

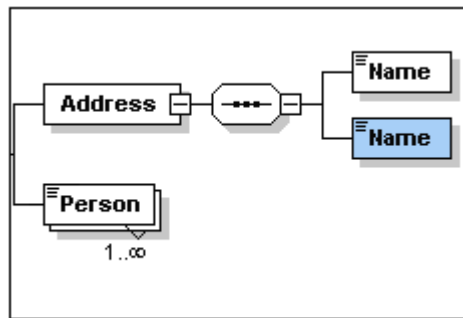
You have added elements using the context menu that appears when you right-click an element or compositor. You can also create elements using drag-and-drop, which is quicker than using menu commands. In this section, you will add more elements to the definition of the `Address` element using drag-and-drop,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

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

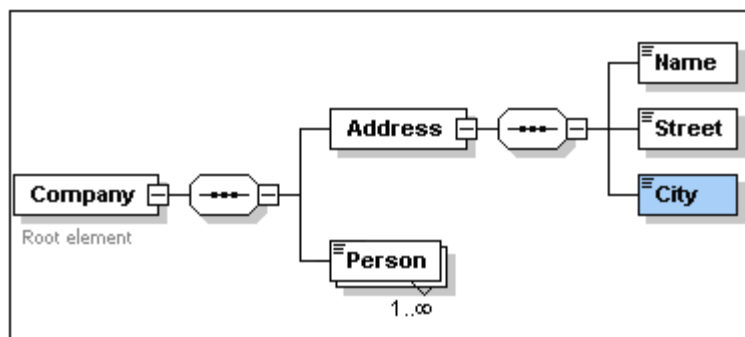
1. Click the `Name` element of the `Address` element, hold down the **Ctrl** key, and drag the element box with the mouse. A small "plus" icon appears in the element box, indicating that you are about to copy the element. A copy of the element together with a connector line also appears, showing where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



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



3. Double-click in the element box, and type in *Street* to change the element name.
4. Use the same method to create a third element called *City*. The content model should now look like this:



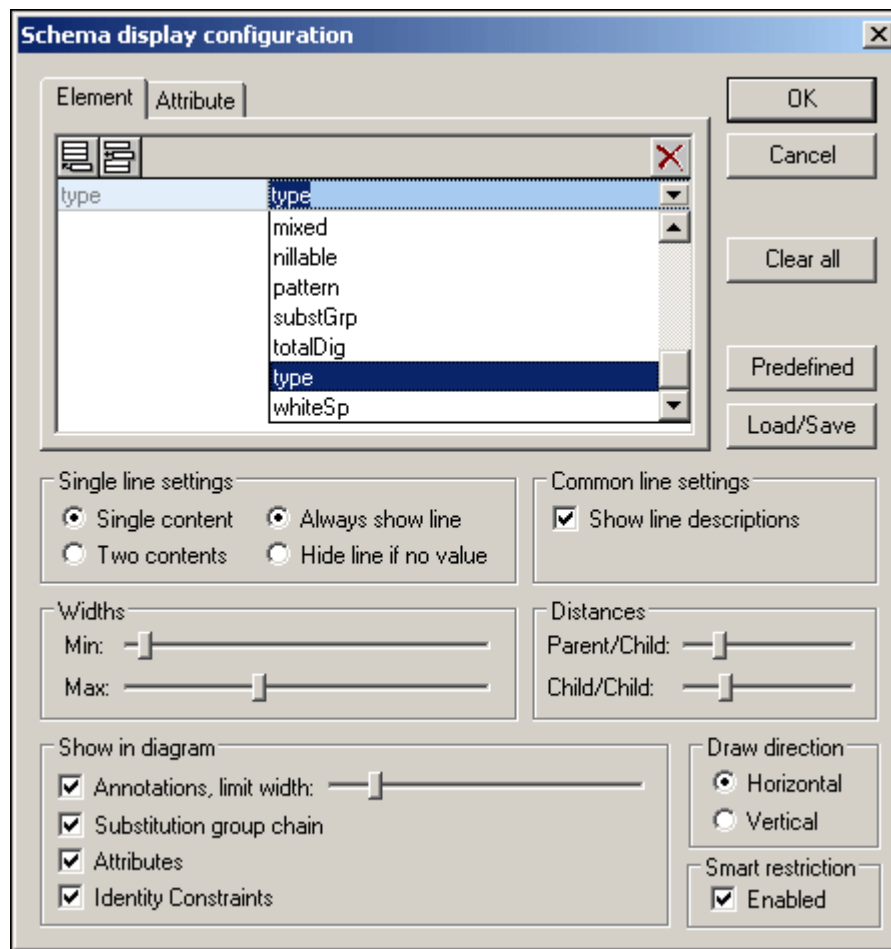
The **Address** element has a sequence of a **Name**, a **Street**, and a **City** element, in that order.

3.2.5 Configuring the Content Model View

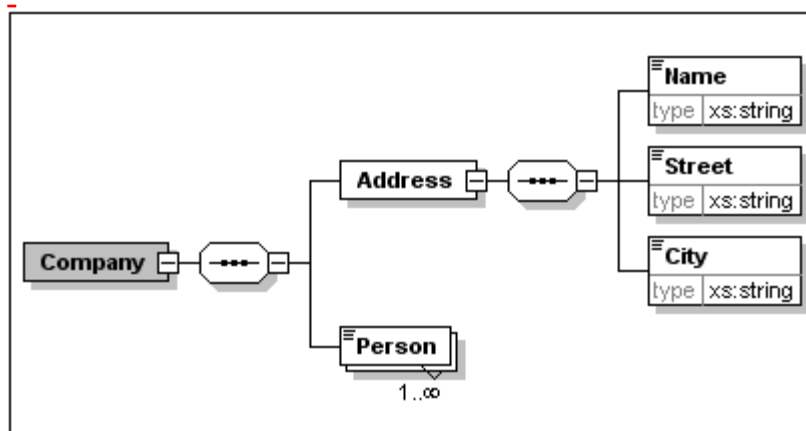
This is a good time to configure the Content Model View. We will configure the Content Model View such that the *type* of the element is displayed for each element.

To configure the Content Model View:

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

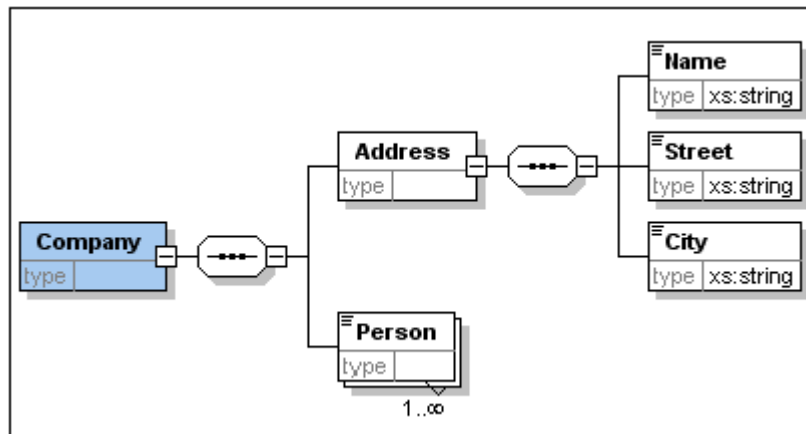


3. Click the **Append**  icon (in the **Element** tab) to add a property descriptor line for each element box.
4. From the dropdown menu, select `type` (or double-click in the line and enter "`type`"). This will cause the data type of each element to be displayed in the Content Model View.
5. In the Single Line Settings pane, select Hide Line If No Value. This hides the description of the datatype in the element box if the element does not have a datatype (for example, if the element is a complex type).



Notice that the type descriptor line appears for the `Name`, `Street`, and `City` elements, which are simple types of type `xs:string`, but not for the complex type elements. This is because the Hide Line If No Value toggle is selected.

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



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

Please note:

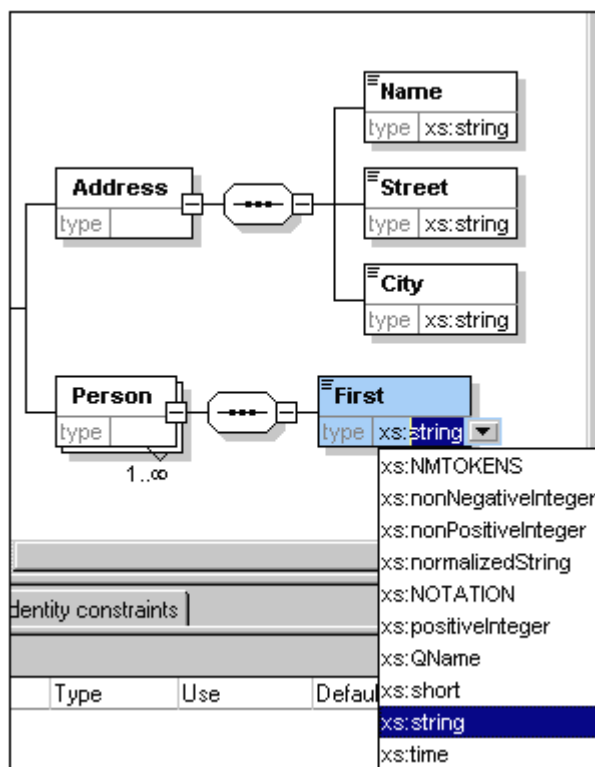
- The property descriptor lines are editable, so values you enter in them become part of the element definition.
- The settings you define in the Schema display configuration dialog apply to the schema documentation output as well as the printer output.

3.2.6 Completing the Basic Schema

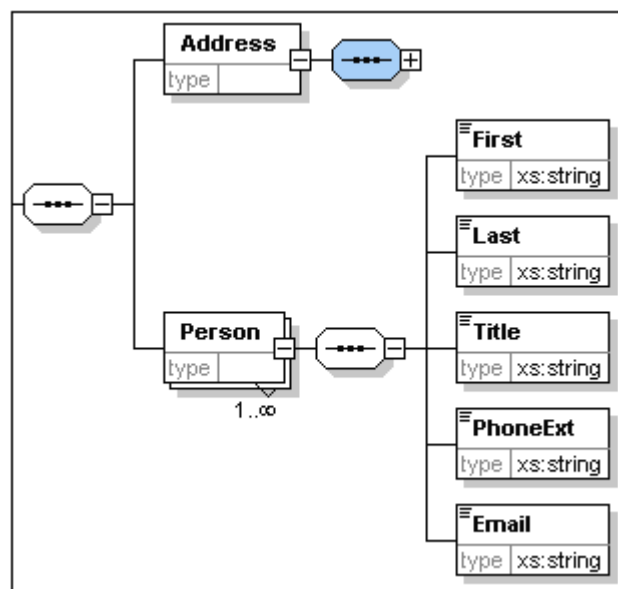
You have defined the content of the `Address` element. Now you need to define the content of the `Person` element. The `Person` element is to contain the following child elements, all of which are simple types: `First`, `Last`, `Title`, `PhoneExt`, and `Email`. All these elements are mandatory except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

To create the content model for `Person`:

1. Right-click the `Person` element to open the context menu, and select **Add Child | Sequence**. This inserts the Sequence compositor.
2. Right-click the Sequence compositor, and select **Add Child | Element**.
3. Enter `First` as the name of the element, and press the **Tab** key. This automatically places the cursor in the `type` field.



4. Select the `xs:string` entry from the dropdown list or enter it into the `type` value field.
5. Use the drag-and-drop method to create four more elements. Name them `Last`, `Title`, `PhoneExt`, and `Email`, respectively.

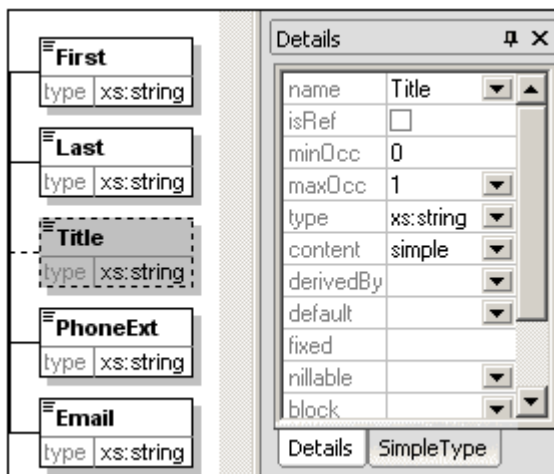


Please note: You can select multiple elements by holding down the **Ctrl** key and clicking each of the required elements. This makes it possible to, e.g., copy several elements at once.

Making an element optional

Right-click the `Title` element and select **Optional** from the context menu. The frame of the

element box changes from solid to dashed; this is a visual indication that an element is optional.

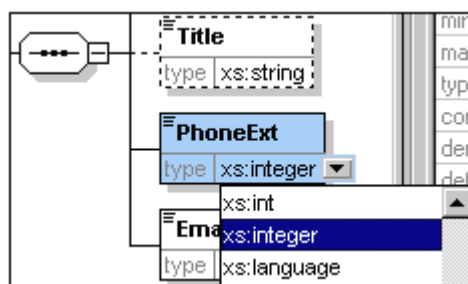


In the Details Entry Helper, you will see that `minOcc=0` and `maxOcc=1`, indicating that the element is optional. Alternatively to using the context menu to make an element optional, you can set `minOcc=0` in order to make the element optional.

Limiting the content of an element

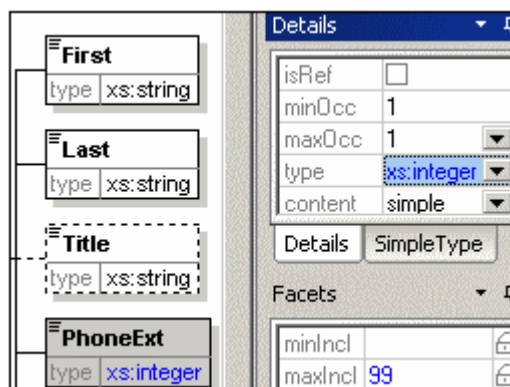
To define the `PhoneExt` element to be of type `xs:integer` and have a maximum of two digits:

1. Double-click in the `type` field of the `PhoneExt` element, and select (or enter) the `xs:integer` entry from the dropdown list.



The items in the Facets Entry Helper change at this point.

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



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

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

Please note:

- Selecting an XML Schema datatype that is a simple type (for example, `xs:string` or `xs:date`), automatically changes the content model to simple in the Details Entry Helper (`content = simple`).
- Adding a compositor to an element (`sequence`, `choice`, or `all`), automatically changes the content model to complex in the Details Entry Helper (`content = complex`).
- The schema described above is available as `AddressFirst.xsd` in the `C:\Documents and Settings\<username>\My Documents\Altova\XMLSpy2012\\Examples\Tutorial` folder of your XMLSpy application folder.

3.3 XML Schemas: Advanced

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

Objective

In this section, you will learn how to:

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

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

Commands used in this section

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



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



Display All Globals. This icon is located at the top left-hand corner of the Content Model View. Clicking the icon switches the view to Schema Overview, which displays all global components.



Append. The Append icon is located at the top left-hand corner of the Schema Overview. Clicking the icon enables you to add a global component.

3.3.1 Working with Complex Types and Simple Types

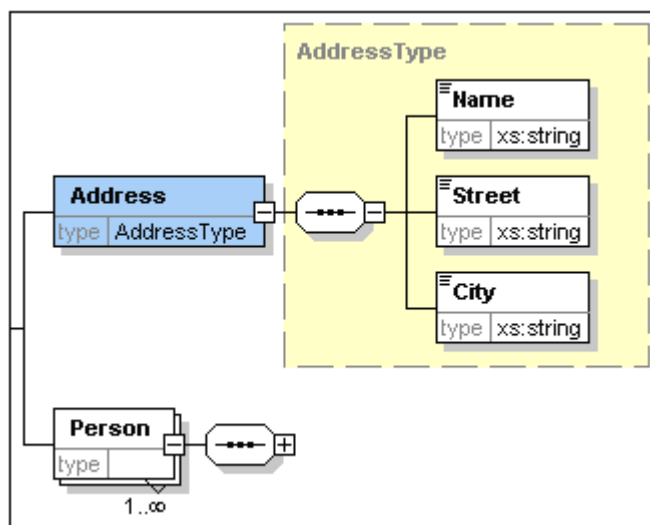
Having defined the content model of an element, you may decide you want to reuse it elsewhere in your schema. The way to do this is by creating that element definition as a global complex type or as a global element. In this section, you will work with global complex types. You will first create a complex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

Creating a global complex type

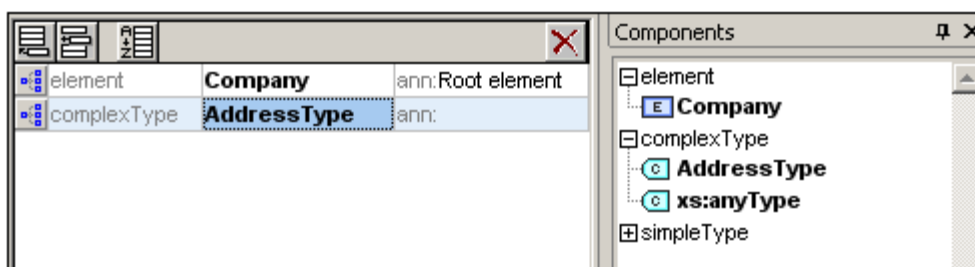
The basic `Address` element that we defined (containing `Name`, `Street`, and `City` elements) can be reused in various address formats. So let us create this element definition as a complex type, which can be reused.

To create a global complex type:

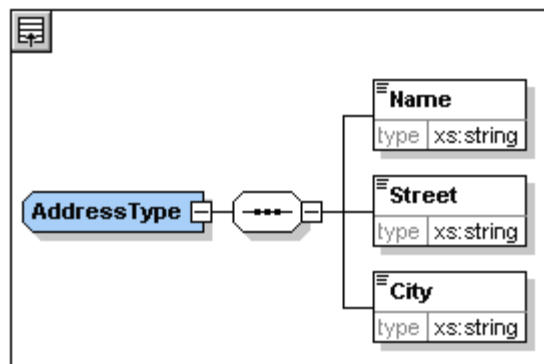
1. In the Content Model View, right-click the `Address` element.
2. In the context menu that now appears, select **Make Global | Complex type**. A global complex type called `AddressType` is created, and the `Address` element in the `Company` content model is assigned this type. The content of the `Address` element is the content model of `AddressType`, which is displayed in a yellow box. Notice that the datatype of the `Address` element is now `AddressType`.



3. Click the Display All Globals  icon. This takes you to the Schema Overview, in which you can view all the global components of the schema.
4. Click the expand icons for the **element** and **complexType** entries in the Components entry helper, to see the respective schema constructs. The Schema Overview now displays two global components: the *Company* element and the complex type *AddressType*. The Components Entry Helper also displays the *AddressType* complex type.



5. Click on the Content Model View icon  of *AddressType* to see its content model (*screenshot below*). Notice the shape of the complex type container.



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

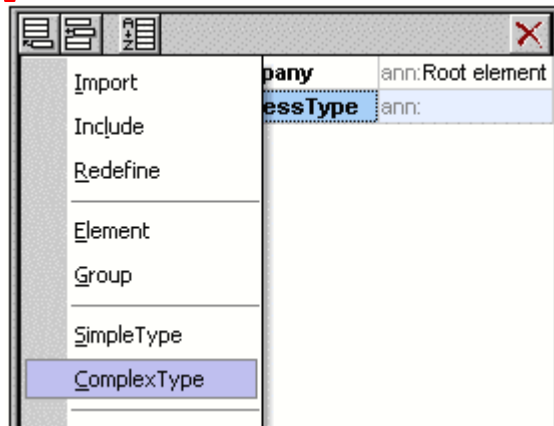
Extending a complex type definition

We now want to use the global *AddressType* component to create two kinds of

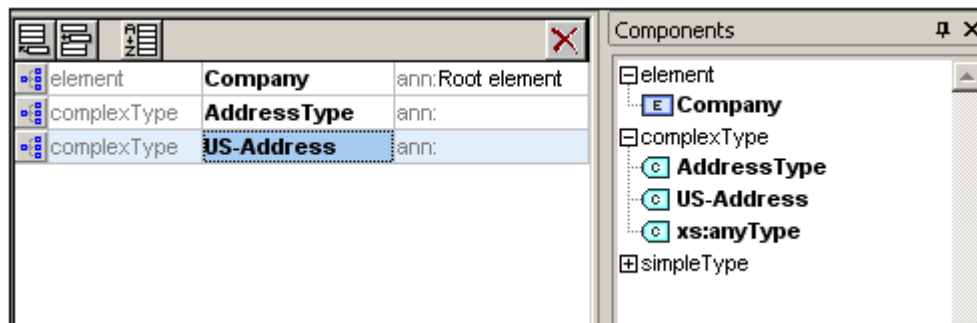
country-specific addresses. For this purpose we will define a new complex type based on the basic `AddressType` component, and then extend that definition.

Do this as follows:

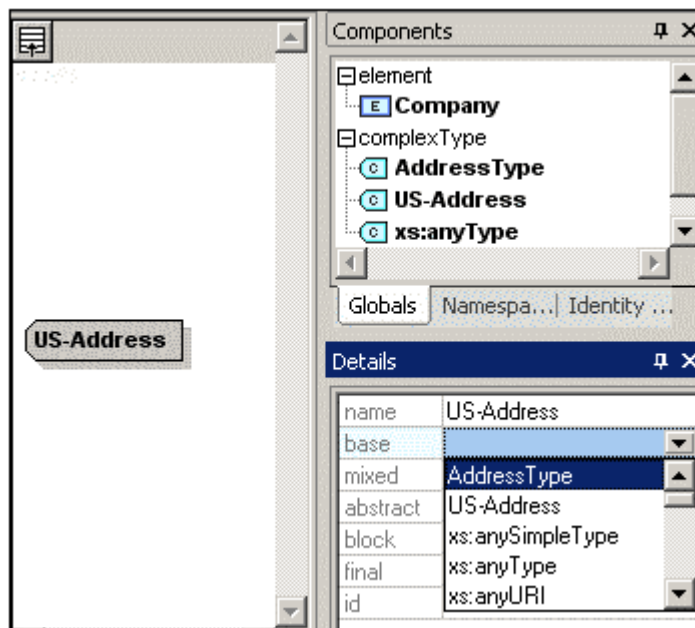
1. Switch to Schema Overview. (If you are in Content Model View, click the Display All Globals icon )
2. Click the Append icon  at the top left of the component window. The following menu opens:



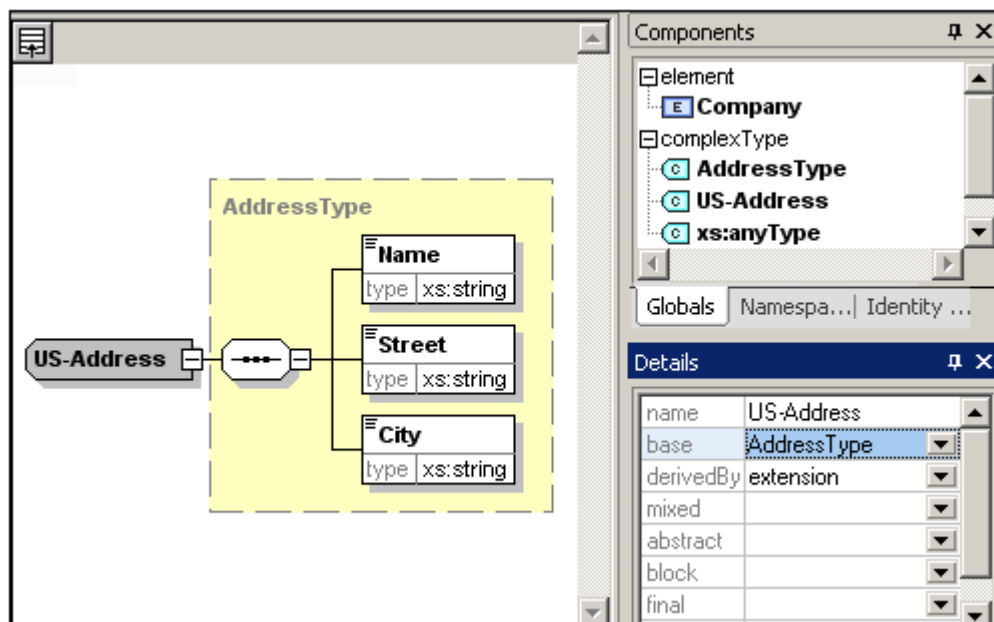
3. Select **ComplexType** from the menu. A new line appears in the component list, and the cursor is set for you to enter the component name.
4. Enter `US-Address` and confirm with **Enter**. (If you forget to enter the hyphen character "-" and enter a space, the element name will appear in red, signalling an invalid character.)



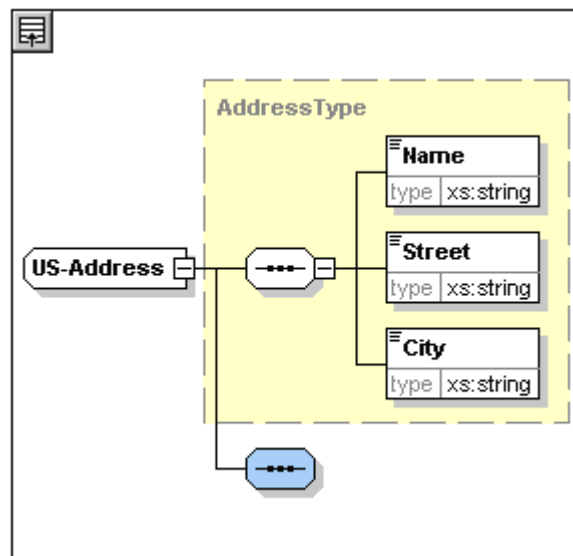
5. Click the Content Model View icon  of `US-Address` to see the content model of the new complex type. The content model is empty (see screenshot below).
6. In the Details entry helper, click the `base` combo box and select the `AddressType` entry.



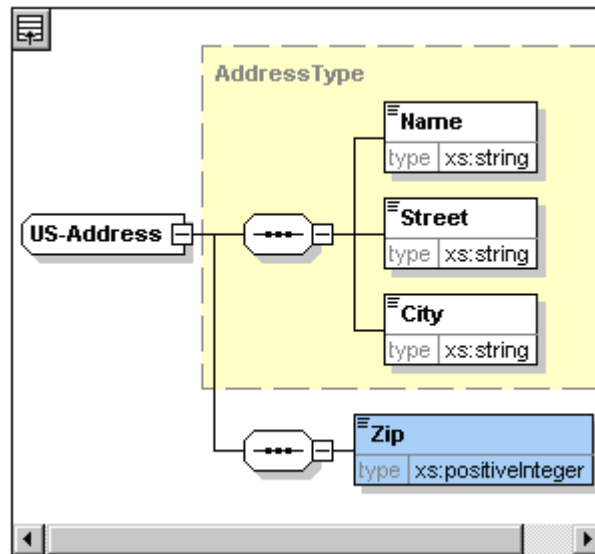
The Content Model View now displays the `AddressType` content model as the content model of `US-Address` (screenshot below).



- Now we can extend the content model of the `US-Address` complex type to take a ZIP Code element. To do this, right-click the `US-Address` component, and, from the context menu that appears, select **Add Child | Sequence**. A new sequence compositor is displayed outside the `AddressType` box (screenshot below). This is a visual indication that this is an extension to the element.



8. Right-click the new sequence compositor and select **Add Child | Element**.
9. Name the newly created element `zip`, and then press the **Tab** key. This places the cursor in the value field of the type descriptor line.
10. Select `xs:positiveInteger` from the dropdown menu that appears, and confirm with **Enter**.



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

Global simple types

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

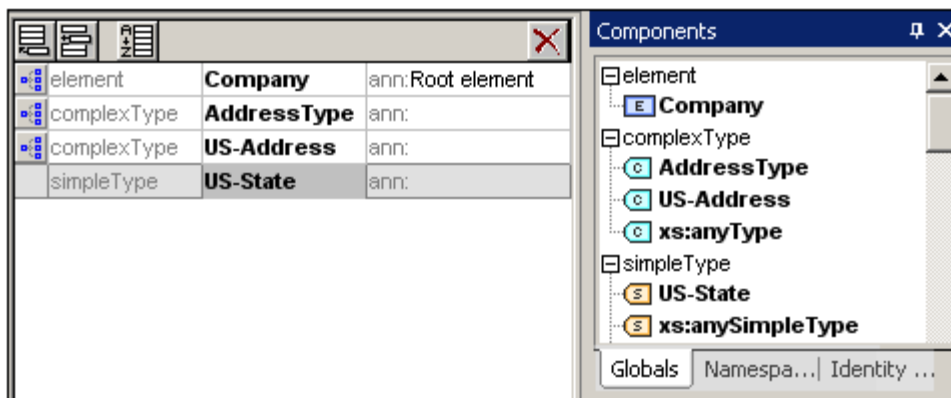
Creating a global simple type

Creating a global simple type consists of appending a new simple type to the list of global

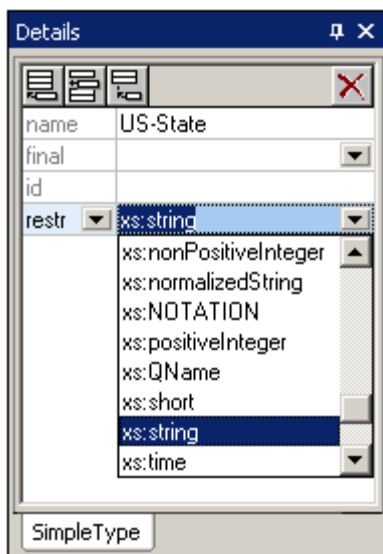
components, naming it, and defining its datatype.

To create a global simple type:

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



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



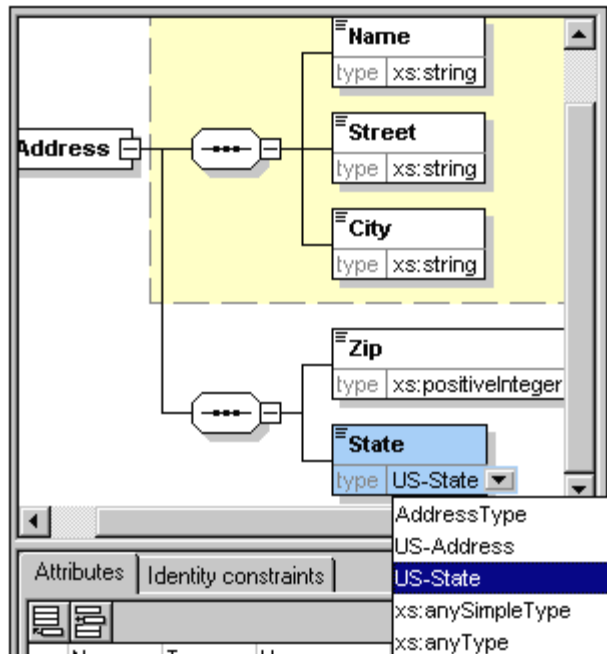
This creates a simple type called `US-State`, which is of datatype `xs:string`. This global component can now be used in the content model of `US-Address`.

Using a global simple type in a content model

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

Do the following:

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



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

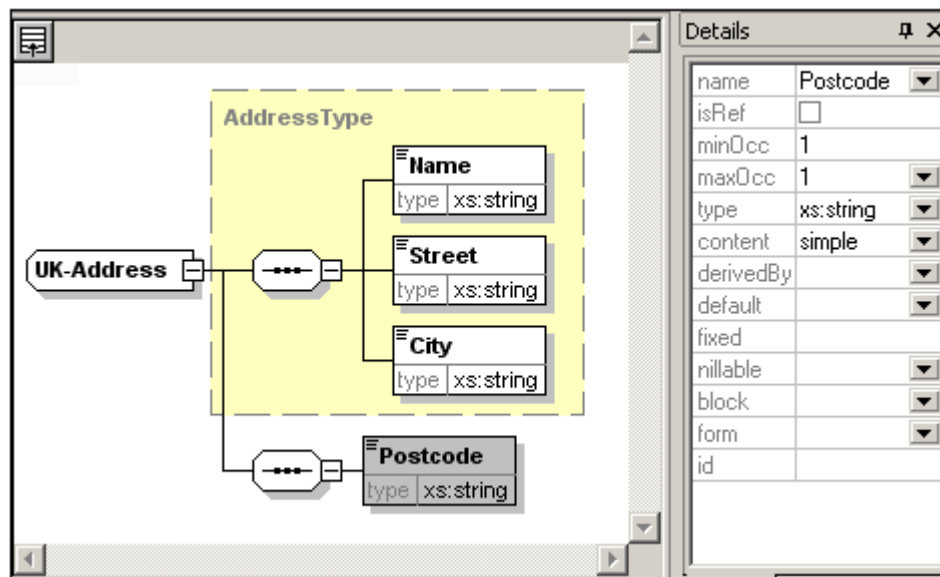
Creating a second complex type based on `AddressType`

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

Do the following:

1. In Schema Overview, create a global complex type called `UK-Address`, and base it on `AddressType` (`base=AddressType`).
2. In the Content Model View of `UK-Address`, add a `Postcode` element and give it a type of `xs:string`.

Your `UK-Address` content model should look like this:

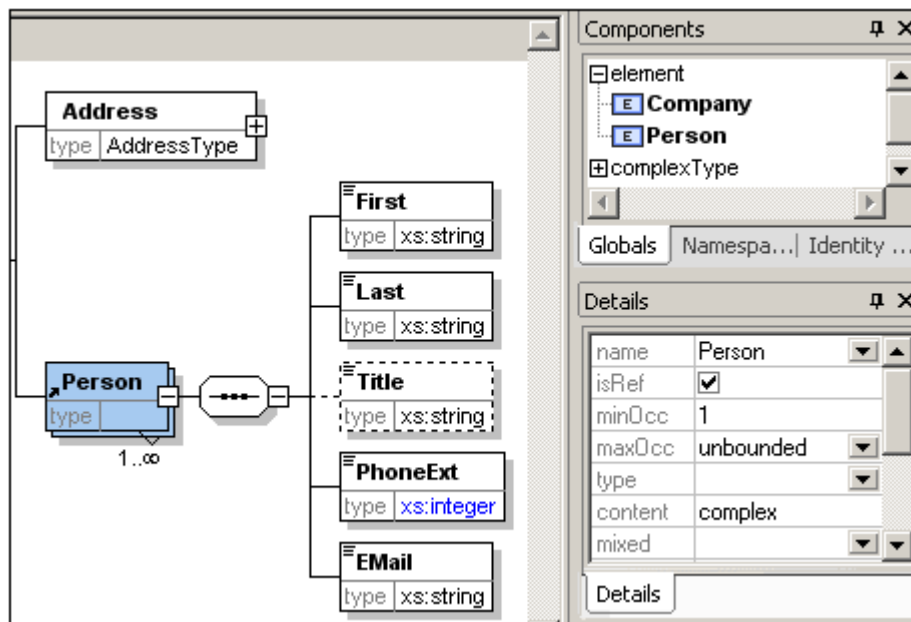


Please note: In this section you created global simple and complex types, which you then used in content model definitions. The advantage of global types is that they can be reused in multiple definitions.

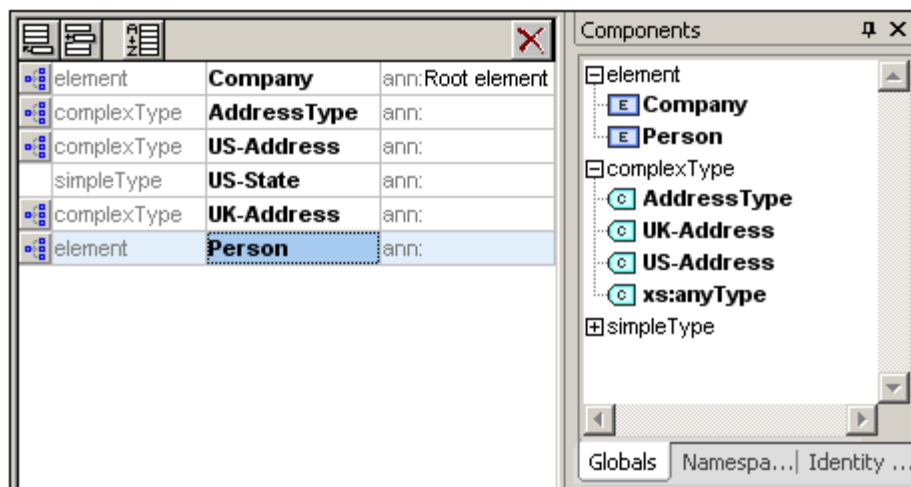
3.3.2 Referencing Global Elements

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

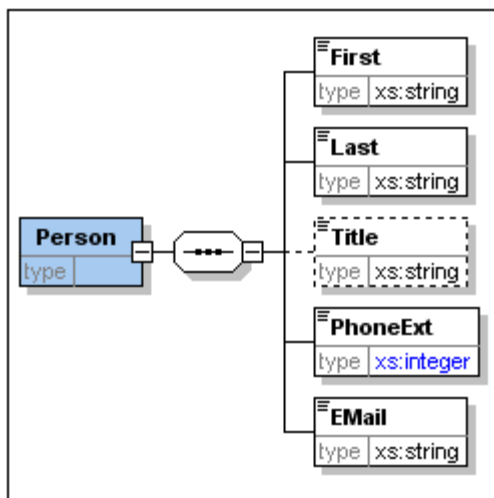
1. Click  (Display All Globals) to switch to Schema Overview.
2. Click the Display Diagram icon  of the `Company` element.
3. Right-click the `Person` element, and select **Make Global | Element**. A small link arrow icon appears in the `Person` element, showing that this element now references the globally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



- Click the Display All Globals icon  to return to Schema Overview. The `Person` element is now listed as a global element. It is also listed in the Components Entry Helper.



5. In the Components Entry Helper, double-click the `Person` element to see the content model of the global `Person` element.



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

Please note:

- An element that references a global element must have the same name as the global element it references.
- A global declaration does not describe where a component is to be used in an XML document. It only describes a content model. It is only when a global declaration is referenced from within another component that its location in the XML document is specified.

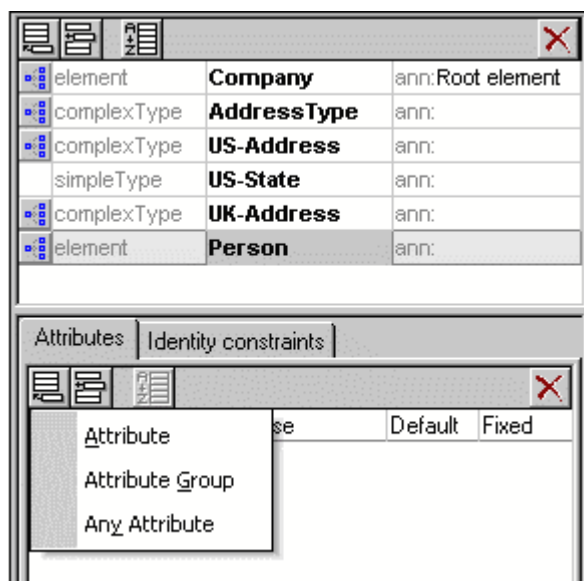
A globally declared element can be reused at multiple locations. It differs from a globally declared complex type in that its content model cannot be modified without also modifying the global element itself. If you change the content model of an element that references a global element, then the content model of the global element will also be changed, and, with it, the content model of all other elements that reference that global element.

3.3.3 Attributes and Attribute Enumerations

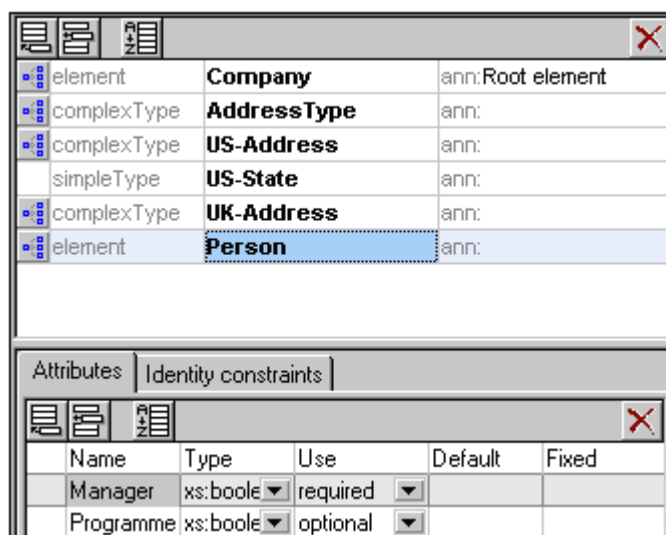
In this section, you will learn how to create attributes and enumerations for attributes.

Defining element attributes

1. In the Schema Overview, click the `Person` element to make it active.
2. Click the Append icon , in the top left of the Attributes/Identity Constraints tab group (in the lower part of the Schema Overview window), and select the Attribute entry.



3. Enter `Manager` as the attribute name in the Name field.
4. Use the Type combo box to select `xs:boolean`.
5. Use the Use combo box to select `required`.



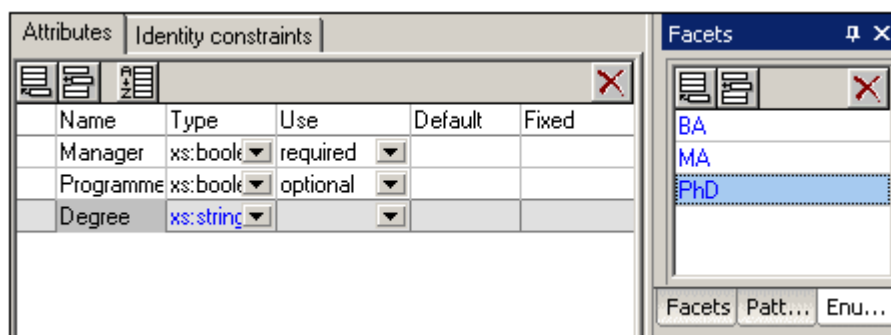
6. Use the same procedure to create a `Programmer` attribute with Type=`xs:boolean` and Use=`optional`.

Defining enumerations for attributes

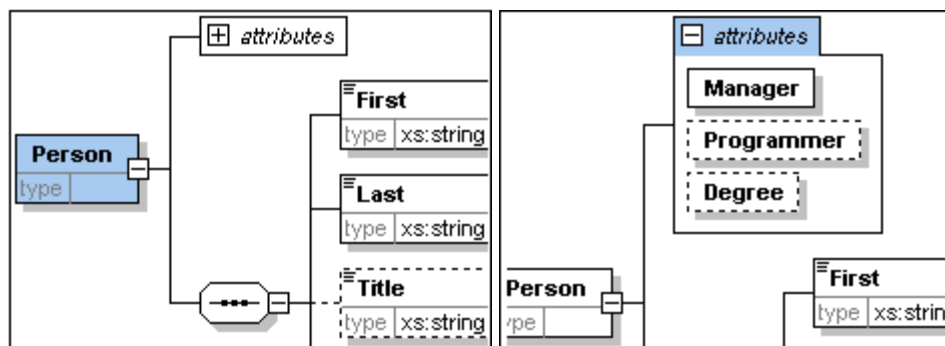
Enumerations are values allowed for a given attribute. If the value of the attribute in the XML instance document is not one of the enumerations specified in the XML Schema, then the document is invalid. We will create enumerations for the `Degree` attribute of the `Person` element.

Do the following:

1. In the Schema Overview, click the `Person` element to make it active.
2. Click the Append icon  in the top left of the Attributes window, and select the **Attribute** entry.
3. Enter `Degree` as the attribute name, and select `xs:string` as its type.
4. With the `Degree` attribute selected, in the Facets Entry Helper, click the **Enumerations** tab (see screenshot).



5. In the **Enumerations** tab, click the Append icon .
6. Enter `BA`, and confirm with **Enter**.
7. Use the same procedure to add two more enumerations: `MA` and `PhD`.
8. Click on the Content Model View icon  of `Person`.



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

9. Click the Display all Globals icon  to return to the Schema Overview.

Saving the completed XML Schema

Please note: Before saving your schema file, rename the `AddressLast.xsd` file that is delivered with XMLSpy to something else (such as `AddressLast_original.xsd`), so as not

to overwrite it.

Save the completed schema with any name you like (**File | Save as**). We recommend you save it with the name `AddressLast.xsd` since the XML file you create in the next part of the tutorial will be based on the `AddressLast.xsd` schema.

3.4 XML Schemas: XMLSpy Features

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

Commands used in this section

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



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

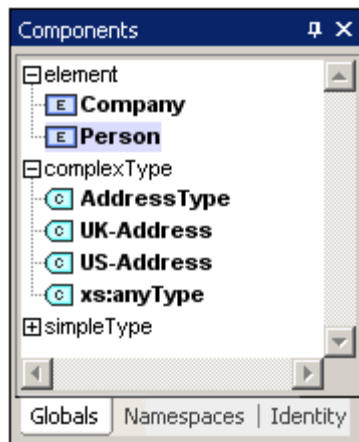
3.4.1 Schema Navigation

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

Displaying the content model of a global component

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

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



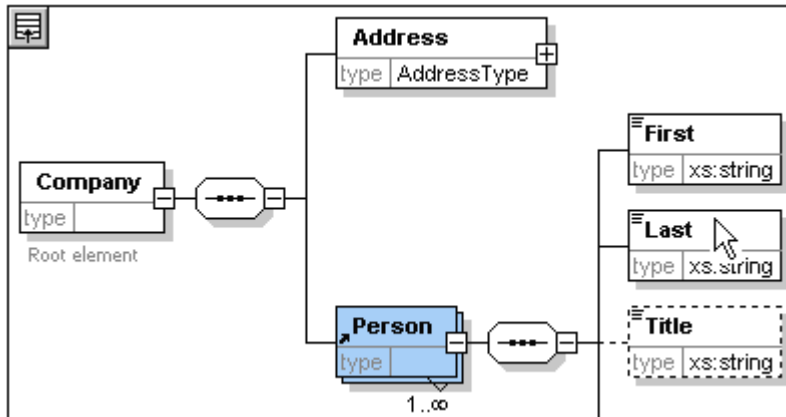
If you double-click any of the other global components (simple type, attribute, attribute group) in the Components Entry Helper, that component will be highlighted in Schema Overview (since they do not have a content model).

In the Components Entry Helper, the double-clicking mechanism works from both the By Type and By Namespace tabs.

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

If a content model contains an element that references a global element, you can go directly to the content model of that global element or to any of its contained components by holding down **Ctrl** and double-clicking the required element.

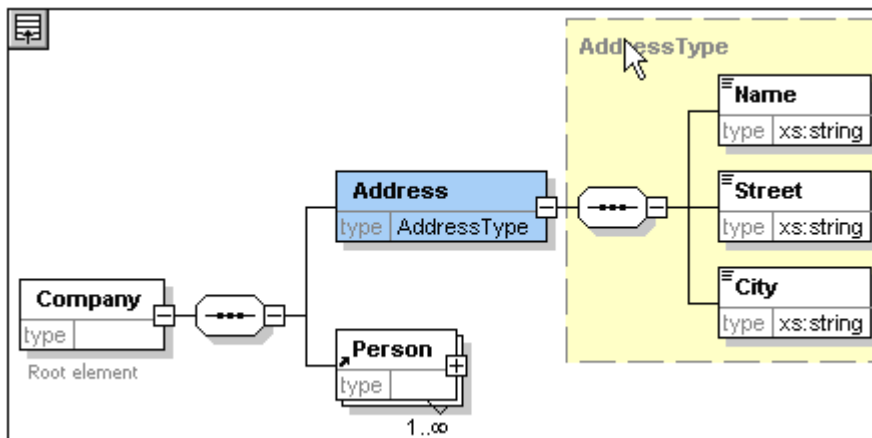
For example, while viewing the `Company` content model, holding down **Ctrl** while double-clicking `Last` opens the `Person` content model and highlights the `Last` element in it.



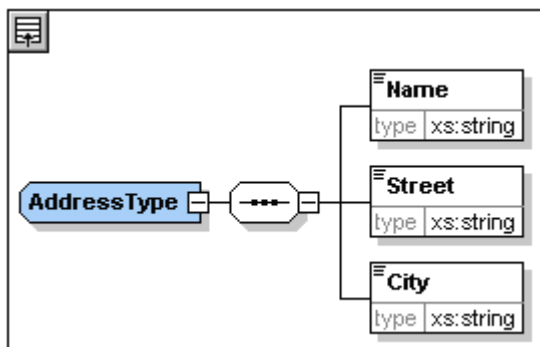
When the `Last` element is highlighted, all its properties are immediately displayed in the relevant entry helpers and information window.

Going to the definition of a complex type

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



This takes you to the Content Model View of the complex type.



Please note: Just as with referenced global elements, you can go directly to an element within the complex type definition by holding down **Ctrl** and double-clicking the required element in the content model that contains the complex type.

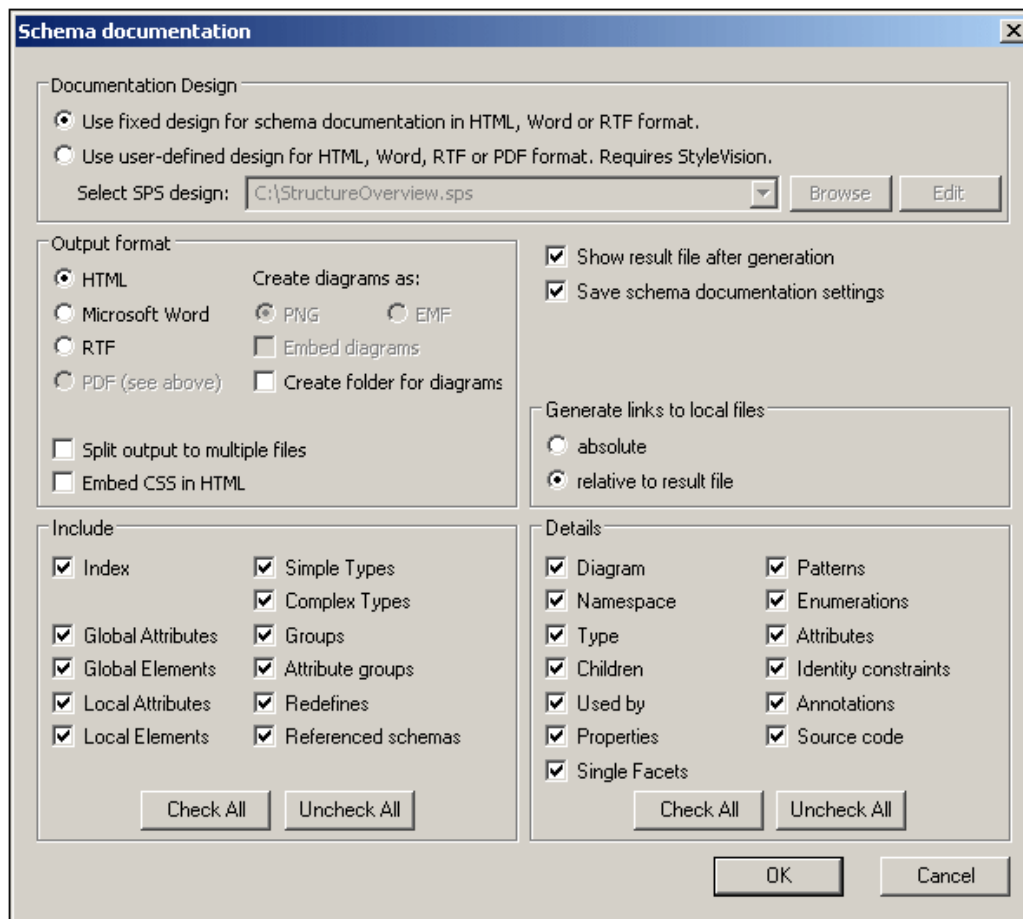
3.4.2 Schema Documentation

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

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

Do the following:

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



2. For the Output Format option, select HTML, and click **OK**.
3. In the Save As dialog, select the location where the file is to be saved and give the file a suitable name (say `AddressLast.html`). Then click the **Save** button.

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

Schema **AddressLast.xsd**

schema location: **C:\Program Files\Altova\XMLSpy2010\Examples\Tutorial\AddressLast.xsd**
 attribute form **unqualified**
 default:
 element form **qualified**
 default:
 targetNamespace: **http://my-company.com/namespace**

Elements [Complex types](#) [Simple types](#)

[Company](#) [AddressType](#) [US-State](#)

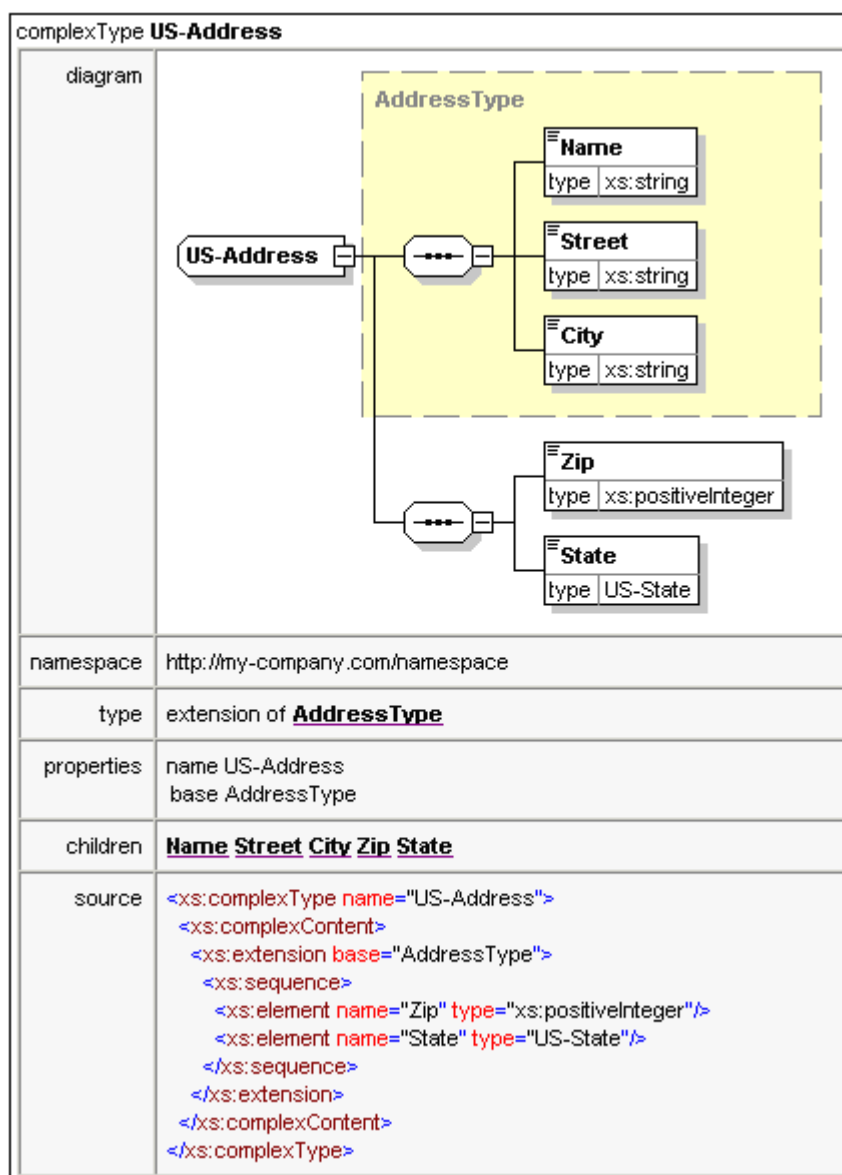
[Person](#) [UK-Address](#)

[US-Address](#)

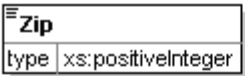
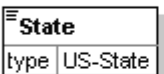
element **Company**

diagram	Root element
namespace	http://my-company.com/namespace
properties	content complex
children	Address Person
annotation	documentation Root element
source	<pre> <xs:element name="Company"> <xs:annotation> <xs:documentation>Root element</xs:documentation> </xs:annotation> <xs:complexType> <xs:sequence> <xs:element name="Address" type="AddressType" /> <xs:element ref="Person" maxOccurs="unbounded" /> </xs:sequence> </xs:complexType> </xs:element> </pre>

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



The diagram above shows how complex types are documented.

element US-Address/Zip	
diagram	
namespace	http://my-company.com/namespace
type	xs:positiveInteger
properties	name Zip isRef 0 content simple
source	<code><xs:element name="Zip" type="xs:positiveInteger"/></code>
element US-Address/State	
diagram	
namespace	http://my-company.com/namespace
type	<u>US-State</u>
properties	name State isRef 0 content simple
source	<code><xs:element name="State" type="US-State"/></code>
simpleType US-State	
namespace	http://my-company.com/namespace
type	xs:string
properties	name US-State
used by	element <u>US-Address/State</u>
source	<code><xs:simpleType name="US-State"> <xs:restriction base="xs:string"/> </xs:simpleType></code>

The diagram above shows how elements and simple types are documented.

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

3.5 XML Documents

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

Objective

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

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

Commands used in this section

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



File | New. Creates a new type of XML file.



View | Text View. Switches to Text View.



View | Grid View. Switches to Enhanced Grid View.



XML | Table | Display as Table. Displays multiple occurrences of a single element type at a single hierarchic level as a table. This view of the element is called the Database/Table View (or simply Table View). The icon is used to switch between the Table View and regular Grid View.



F7. Checks for well-formedness.



F8. Validates the XML document against the associated DTD or Schema.



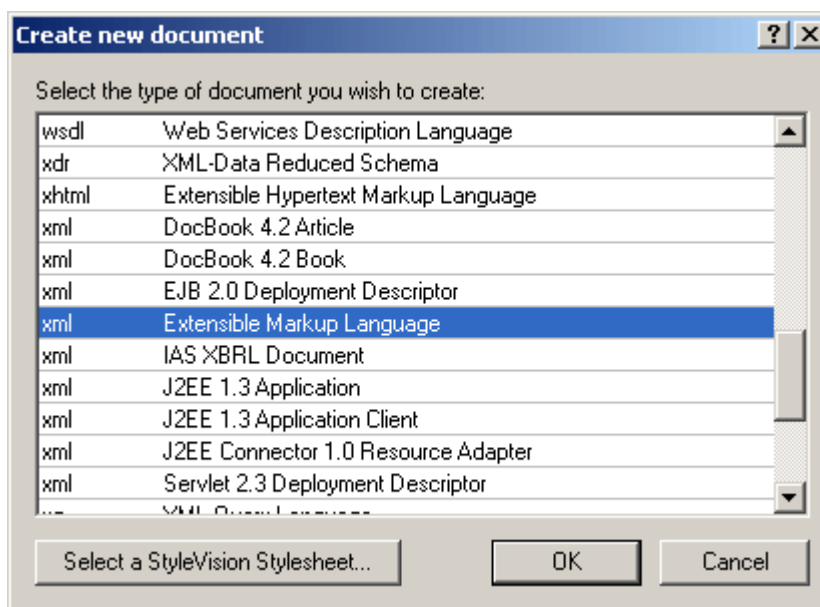
Opens the associated DTD or XML Schema file.

3.5.1 Creating a New XML File

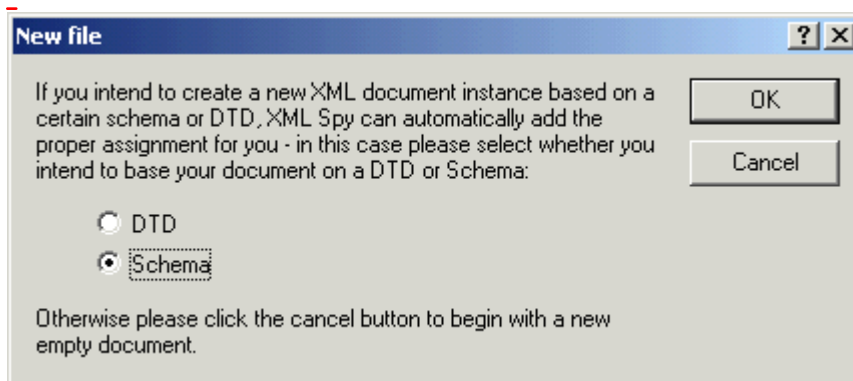
When you create a new XML file in XMLSpy, you are given the option of basing it on a schema (DTD or XML Schema) or not. In this section you will create a new file that is based on the `AddressLast.xsd` schema you created earlier in the tutorial.

To create the new XML file:

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

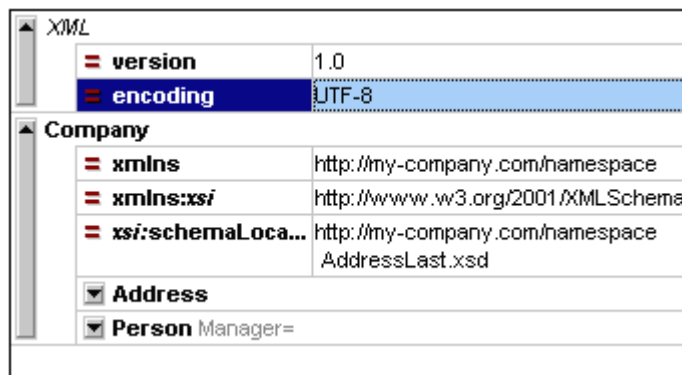


2. Select the `Extensible Markup Language` entry (or generic XML document entry) from the dialog, and confirm with **OK**. A prompt appears, asking if you want to base the XML document on a DTD or Schema.

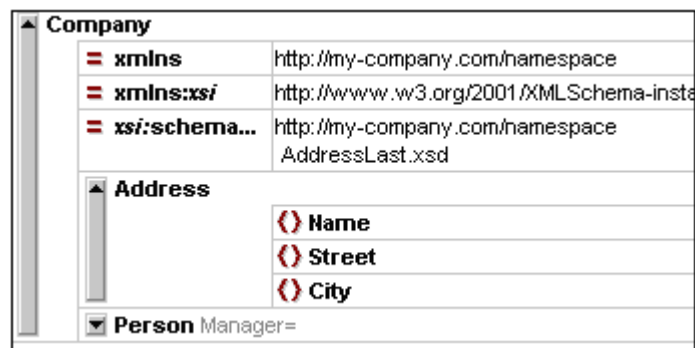


3. Click the Schema radio button, and confirm with **OK**. A further dialog appears, asking you to select the schema file your XML document is to be based on.
4. Use the Browse or Window buttons to find the schema file. The Window button lists all files open in XMLSpy and projects. Select `AddressLast.xsd`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

-

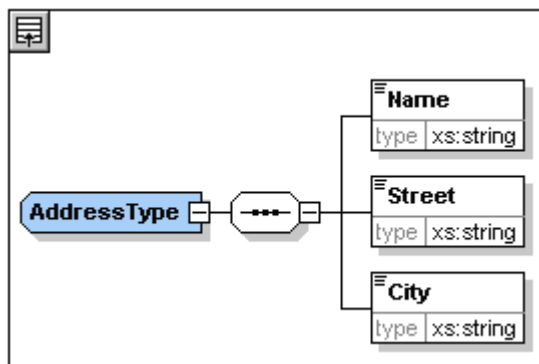


7. Click on the  icon next to **Address**, to view the child elements of **Address**. Your document should look like this:



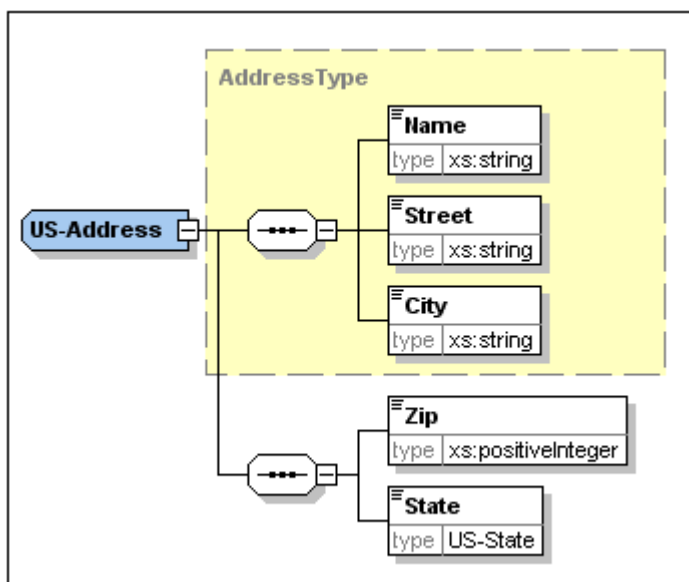
3.5.2 Specifying the Type of an Element

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



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

-



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

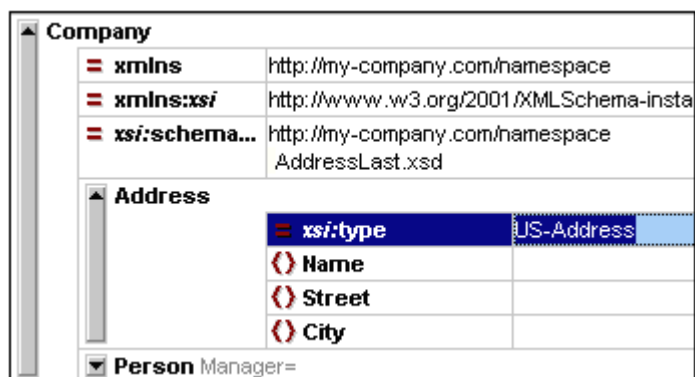
Do the following:

1. In the XML document, right-click the `Name` element, and select **Insert | Attribute** from the context menu.



An attribute field is added to the `Address` element.

2. Ensure that `xsi:type` is entered as the name of the attribute (screenshot below).
3. Press **Tab** to move into the next (value) field.



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

Please note: The `xsi` prefix allows you to use special XML Schema related commands in your XML document instance. Notice that the namespace for the `xsi` prefix was automatically

added to the document element when you assigned a schema to your XML file. In the above case, you have specified a type for the `Address` element. See the [XML Schema specification](#) for more information.

3.5.3 Entering Data in Grid View

You can now enter data into your XML document.

Do the following:

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

Company	
xm:ns	http://my-company.com/namespace
xm:ns:xsi	http://www.w3.org/2001/XMLSchema-instance
xsi:schema...	http://my-company.com/namespace/AddressLast.xsd
Address	
xsi:type	US-Address
Name	US dependency
Street	
City	
Person Manager=	

2. Use the same method to enter a `Street` and `City` name (for example, `Noble Ave` and `Dallas`).
3. Click the `Person` element and press **Delete** to delete the `Person` element. (We will add it back in the next section of the tutorial.) After you do this, the entire `Address` element is highlighted.
4. Click on any child element of the `Address` element to deselect all the child elements of `Address` except the selected element. Your XML document should look like this:

Company	
xm:ns	http://my-company.com/namespace
xm:ns:xsi	http://www.w3.org/2001/XMLSchema-instance
xsi:schema...	http://my-company.com/namespace/AddressLast.xsd
Address	
xsi:type	US-Address
Name	US dependency
Street	Noble Ave
City	Dallas

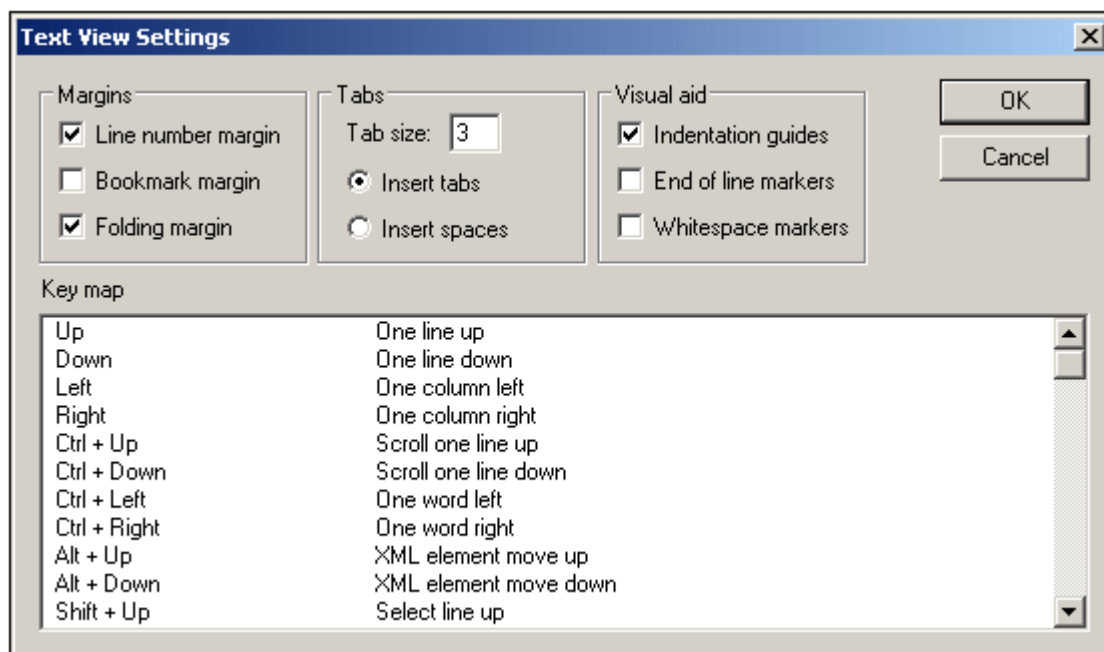
3.5.4 Entering Data in Text View

Text View is ideal for editing the actual data and markup of XML files because of its DTD/XML Schema-related intelligent editing features.

Structural layout features

In addition, Text View has a number of viewing and structural editing features that make editing

large sections of text easy. These features can be switched on and off in the Text View Settings dialog (**View | Text View Settings**, *screenshot below*).



The following margins in Text View can be switched on and off:

- Line number margins
- Bookmark margins, in which individual lines can be highlighted with a marker
- Source folding margins, which contain icons to expand and collapse the display of elements

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

The bookmark feature is useful for setting up markers in your document. To insert a bookmark, use the command **Edit | Insert/Remove Bookmark**. Once bookmarks have been inserted you can navigate these bookmarks using commands in the **Edit** menu.

The screenshot below shows the current XML file in Text View with all structural editing features enabled. For the sake of clarity, none of the line numbers, indentation guides, etc., will be shown in Text View in rest of this tutorial. Please see the User Manual for more information on Text View.

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  = <Company xmlns="http://my-company.com/narr
3    C:\PROGRA~1\Altova\XMLSPY2004\Examples\
4  = <Address xsi:type="US-Address">
5      <Name>US dependency</Name>
6      <Street>Noble Ave</Street>
7      <City>Dallas</City>
8  </Address>

```

Editing in Text View

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

Do the following:

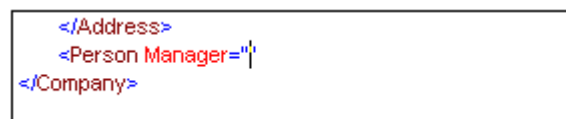
1. Select the menu item **View | Text view**, or click on the **Text** tab. You now see the XML document in its text form, with syntax coloring.



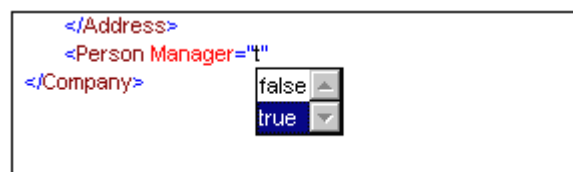
2. Place the text cursor after the end tag of the Address element, and press **Enter** to add a new line.
3. Enter the less-than angular bracket **<** at this position. A dropdown list of all elements allowed at that point (according to the schema) is displayed. Since only the `Person` element is allowed at this point, it will be the only element displayed in the list.



4. Select the `Person` entry. The `Person` element, as well as its attribute `Manager`, are inserted, with the cursor inside the value-field of the `Manager` attribute.



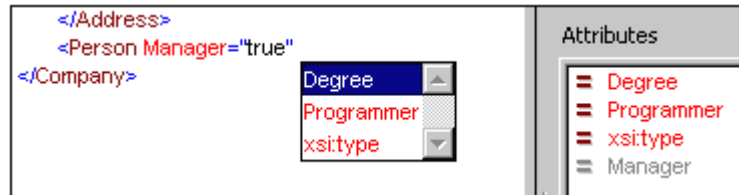
5. From the dropdown list for the `Manager` attribute, select `true`.



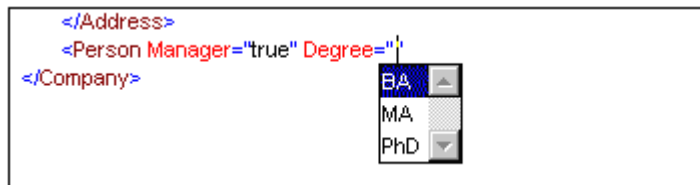
Press **Enter** to insert the value `true` at the cursor position.

6. Move the cursor to the end of the line (using the **End** key if you like), and press the space bar. This opens a dropdown list, this time containing a list of attributes allowed at

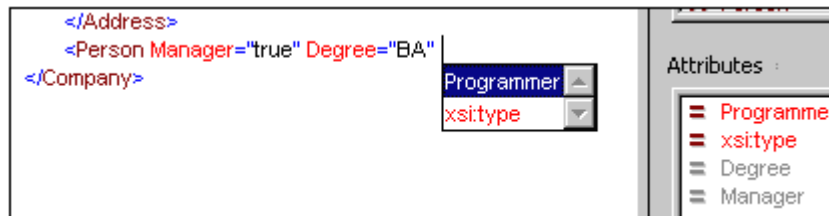
that point. Also, in the Attributes Entry Helper, the available attributes are listed in red. The Manager attribute is grayed out because it has already been used.



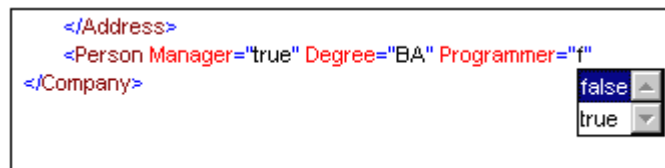
7. Select `Degree` with the Down arrow key, and press **Enter**. This opens another list box, from which you can select one of the predefined enumerations (`BA`, `MA`, or `PhD`).



8. Select `BA` with the Down arrow key and confirm with **Enter**. Then move the cursor to the end of the line (with the **End** key), and press the space bar. `Manager` and `Degree` are now grayed out in the Attributes Entry Helper.



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



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

```

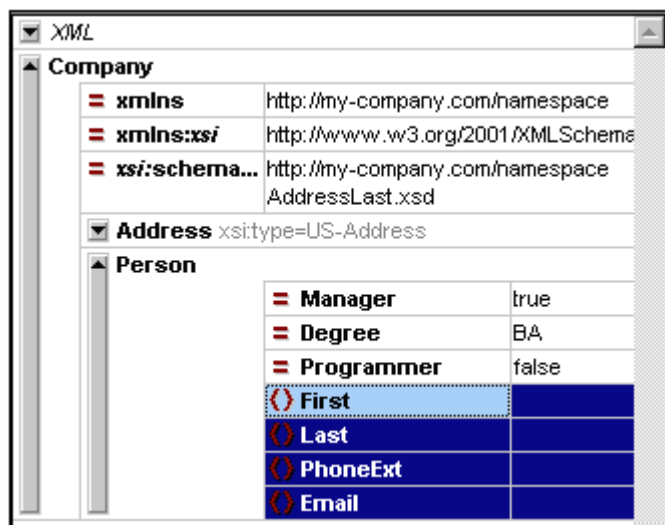
<?xml version="1.0" encoding="UTF-8"?>
<Company xmlns="http://my-company.com/namespace"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://my-company.com/namespace
AddressLast.xsd">
  <Address xsi:type="US-Address">
    <Name>US dependency</Name>
    <Street>Noble Ave</Street>
    <City>Dallas</City>
  </Address>
  <Person Manager="true" Degree="BA" Programmer="false">
    <First></First>
    <Last></Last>
    <PhoneExt></PhoneExt>
    <Email></Email>
  </Person>
</Company>

```

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

Switching to Grid View

To switch to Grid View, select the menu item **View | Grid View**, or click the **Grid** tab. The newly added child elements of `Person` are highlighted.



Now let us validate the document and correct any errors that the validation finds.

3.5.5 Validating the Document

XMLSpy provides two evaluations of the XML document:

- A well-formedness check
- A validation check

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

Checking well-formedness

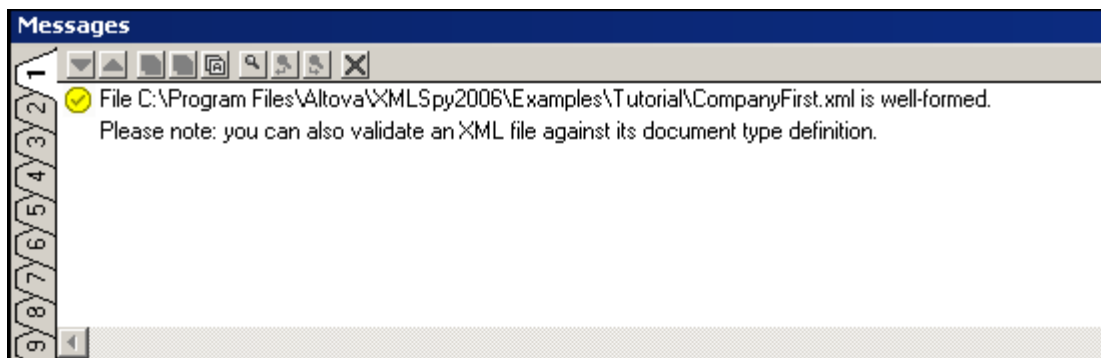
An XML document is well-formed if starting tags match closing tags, elements are nested

correctly, there are no misplaced or missing characters (such as an entity without its semi-colon delimiter), etc.

You can do a well-formedness check in any editing view. Let us select Text View. To do a well-formedness check, select the menu option **XML | Check well-formedness**, or press the

F7 key, or click . A message appears in the Messages window at the bottom of the Main Window saying the document is well-formed.

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



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

Checking validity

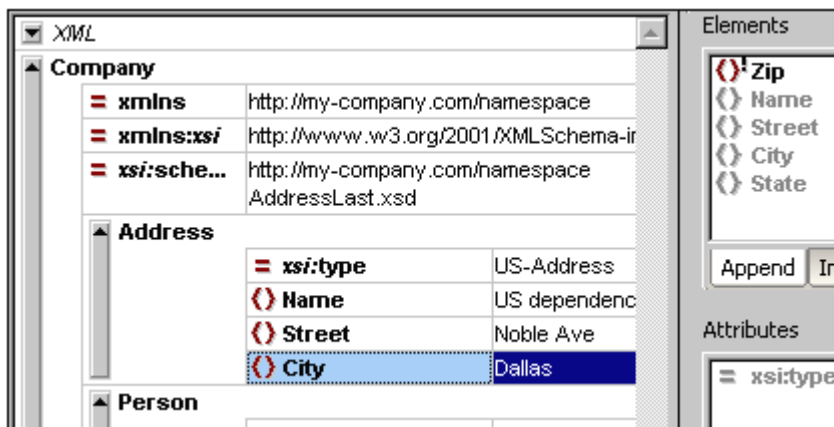
An XML document is valid according to a schema if it conforms to the structure and content specified in that schema.

To check the validity of your XML document, first select Grid View, then select the menu option

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

Fixing the invalid document

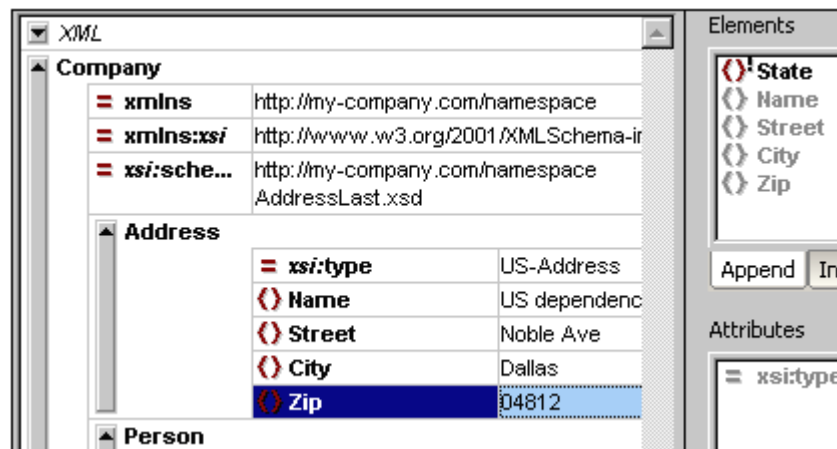
The point at which the document becomes invalid is highlighted, in this case the `City` element.



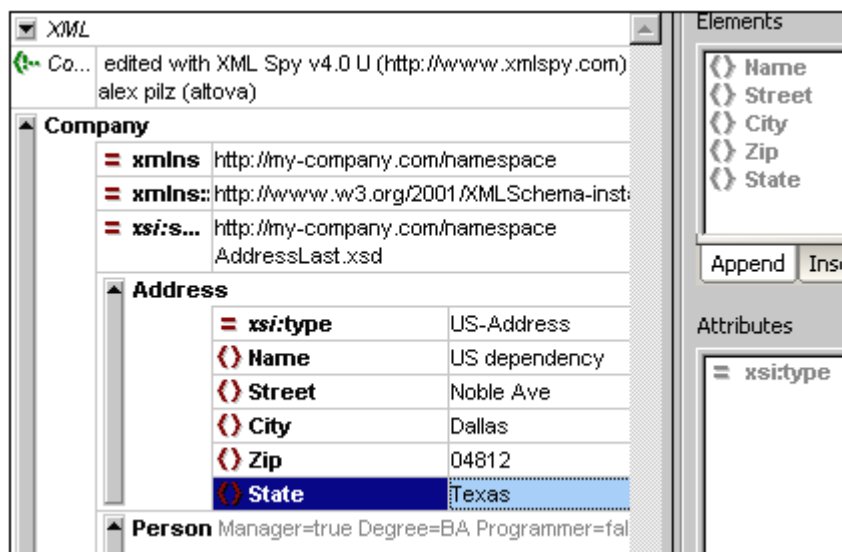
Now look at the Elements Entry Helper (at top right). Notice that the `zip` element is prefixed with an exclamation mark, which indicates that the element is mandatory in the current context.

To fix the validation error:

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



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

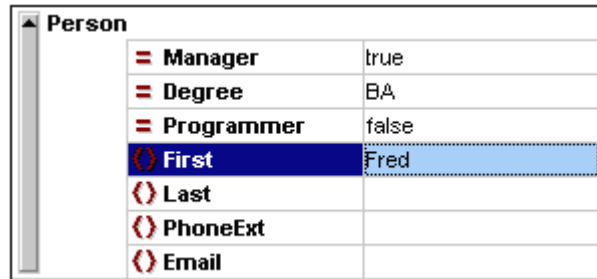


Completing the document and revalidating

Let us now complete the document (enter data for the `Person` element) before revalidating.

Do the following:

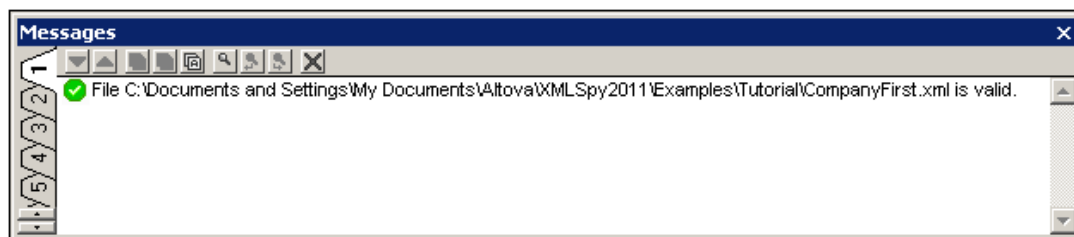
1. Click the value field of the element `First`, and enter a first name (say `Fred`). Then press **Enter**.



2. In the same way enter data for all the child elements of `Person`, that is, for `Last`, `PhoneExt`, and `Email`. Note that the value of `PhoneExt` must be an integer with a maximum value of 99 (since this is the range of allowed `PhoneExt` values you defined in your schema). Your XML document should then look something like this in Grid View:

Company	
= xmlns	http://my-company.com/namespace
= xmlns:xsi	http://www.w3.org/2001/XMLSchema-instance
= xsi:schemaLoca...	http://my-company.com/namespace C:\PROGRA~1\Altova\XMLSpy2006\Examples\Tutorial\AddressLast.xsd
Address	
= xsi:type	US-Address
Ⓜ Name	US Dependency
Ⓜ Street	Noble Ave.
Ⓜ City	Dallas
Ⓜ Zip	04812
Ⓜ State	Texas
Person	
= Manager	true
= Degree	BA
= Programmer	false
Ⓜ First	Fred
Ⓜ Last	Smith
Ⓜ PhoneExt	22
Ⓜ Email	Smith@work.com

3. Click  again to check if the document is valid. A message appears in the Messages window stating that the file is valid. The XML document is now valid against its schema.



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

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

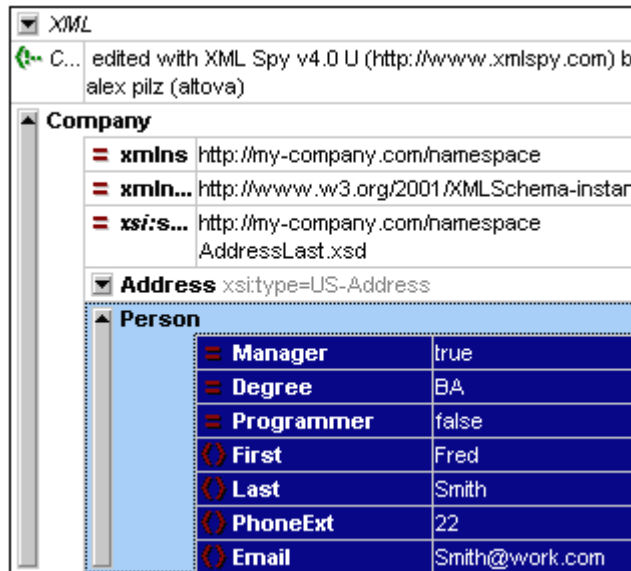
3.5.6 Adding Elements and Attributes

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

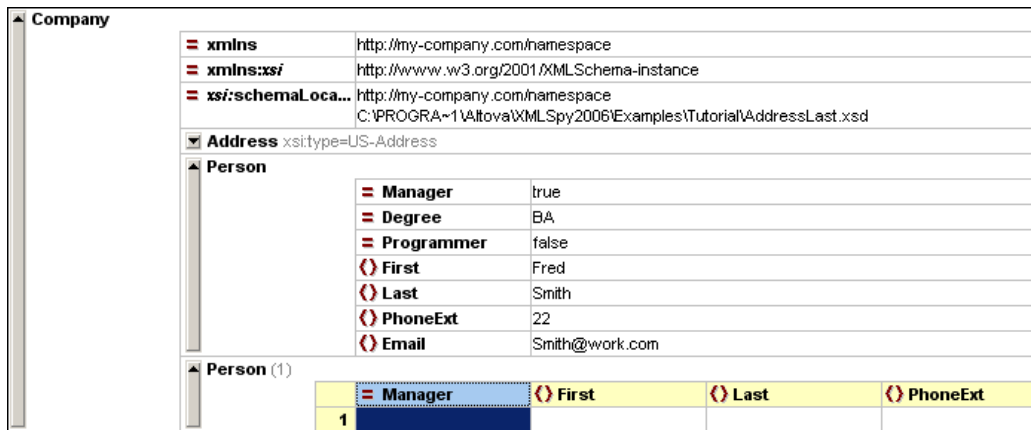
To add a new `Person` element:

1. Click the gray sidebar to the left of the `Address` element to collapse the `Address` element. This clears up some space in the view.
2. Select the entire `Person` element by clicking on or below the `Person` element text in

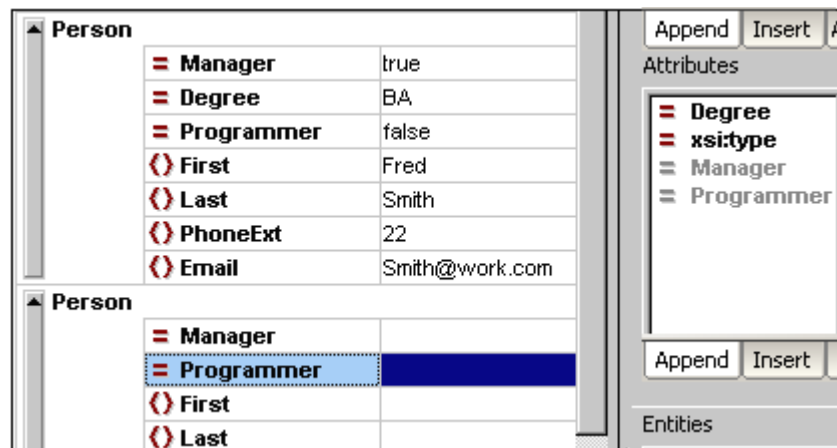
Grid View. Notice that the `Person` element is now available in the **Append** tab of the Elements Entry Helper.



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



- Switch to Grid View and then press **F12** to switch the new `Person` element from Table View to normal Grid View.
- Click on the `Manager` attribute of the new `Person` element. Take a look at the Attributes Entry Helper. The `Manager` entry is grayed out because it has already been entered. Also look at the Info Window, which now displays information about the `Manager` attribute. It is a required attribute and has therefore been added. The `Programmer` attribute has not been added.
- In the **Append** tab of the Attributes Entry Helper, double-click the `Programmer` entry. This inserts an empty `Programmer` attribute after the `Manager` attribute.



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

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

3.5.7 Editing in Database/Table View

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

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

Advantages of Table View

Table View provides the following advantages:

- You can drag-and-drop column headers to reposition the columns relative to each other. This means that, in the actual XML document, the relative position of child elements or attributes is modified for all the element occurrences that correspond to the rows of the table.
- Tables can be sorted (in ascending or descending order) according to the contents of any column using **XML | Table | Ascending Sort** or **Descending Sort**.
- Additional rows (i.e., element occurrences) can be appended or inserted using **XML | Table | Insert Row**.
- You can copy-and-paste **structured data** to and from third party products
- The familiar intelligent editing feature is active in Table View also.

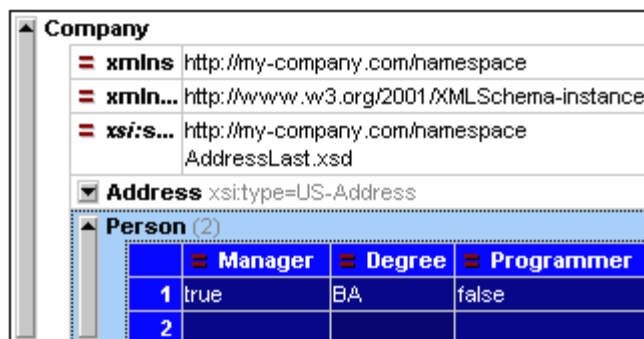
Displaying an element type as a Table

To display the `Person` element type as a table:

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

▼ Address xsi:type=US-Address	
▲ Person	
≡ Manager	true
≡ Degree	BA
≡ Programmer	false
⌂ First	Fred
⌂ Last	Smith
⌂ PhoneExt	22
⌂ Email	Smith@work.com
▲ Person	
≡ Manager	
≡ Programmer	
⌂ First	
⌂ Last	
⌂ PhoneExt	

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



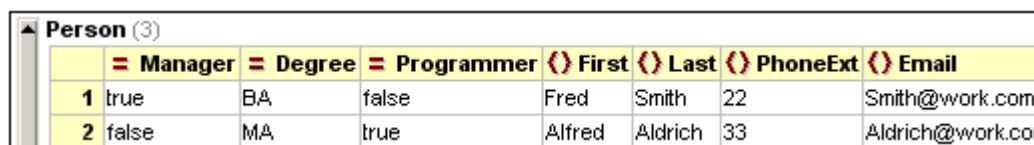
	Manager	Degree	Programmer
1	true	BA	false
2			

3. Select the menu option **View | Optimal widths**, or click the Optimal Widths icon, , to optimize the column widths of the table.

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

Entering content in Table View

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



	Manager	Degree	Programmer	First	Last	PhoneExt	Email
1	true	BA	false	Fred	Smith	22	Smith@work.com
2	false	MA	true	Alfred	Aldrich	33	Aldrich@work.com

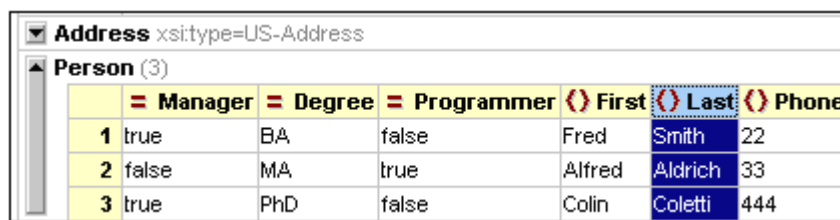
Please note: The Entry Helpers are active also for the elements and attributes displayed as a table. Double-clicking the `Person` entry in the Elements Entry Helper, for example, would add a new row to the table (i.e., a new occurrence of the `Person` element).

Copying XML data to and from third party products

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

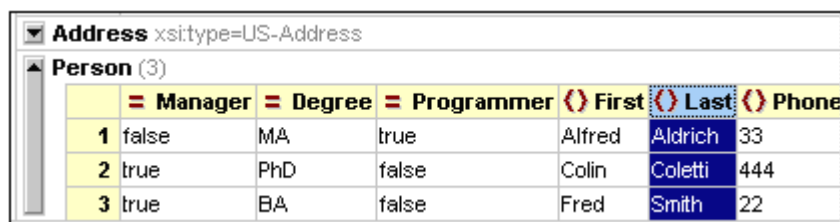
Do the following:

1. Click on the row label `1`, hold down the **Ctrl** key, and click on row label `2`. This selects both rows of the table.



	= Manager	= Degree	= Programmer	First	Last	Phone
1	true	BA	false	Fred	Smith	22
2	false	MA	true	Alfred	Aldrich	33
3	true	PhD	false	Colin	Coletti	444

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



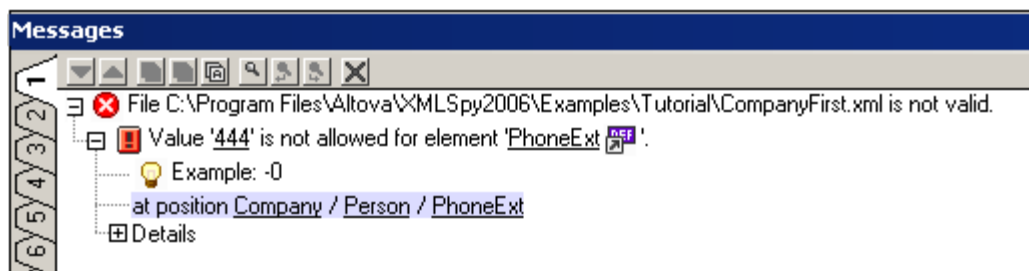
	= Manager	= Degree	= Programmer	First	Last	Phone
1	false	MA	true	Alfred	Aldrich	33
2	true	PhD	false	Colin	Coletti	444
3	true	BA	false	Fred	Smith	22

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

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

Expand "Details" to see that `PhoneExt` is not valid because it is not less than or equal to the maximum value of 99.

Please Note: You can click on the links in the error message to jump to the spot in the XML file where the error was found.



Since the value range we set for phone extension numbers does not cover this extension number, we have to modify the XML Schema so that this number is valid. You will do this in the next section.

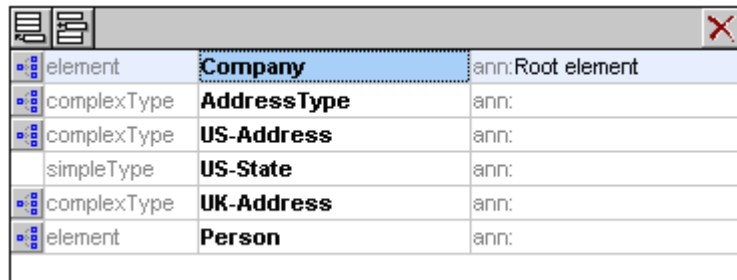
3.5.8 Modifying the Schema

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

Do the following:

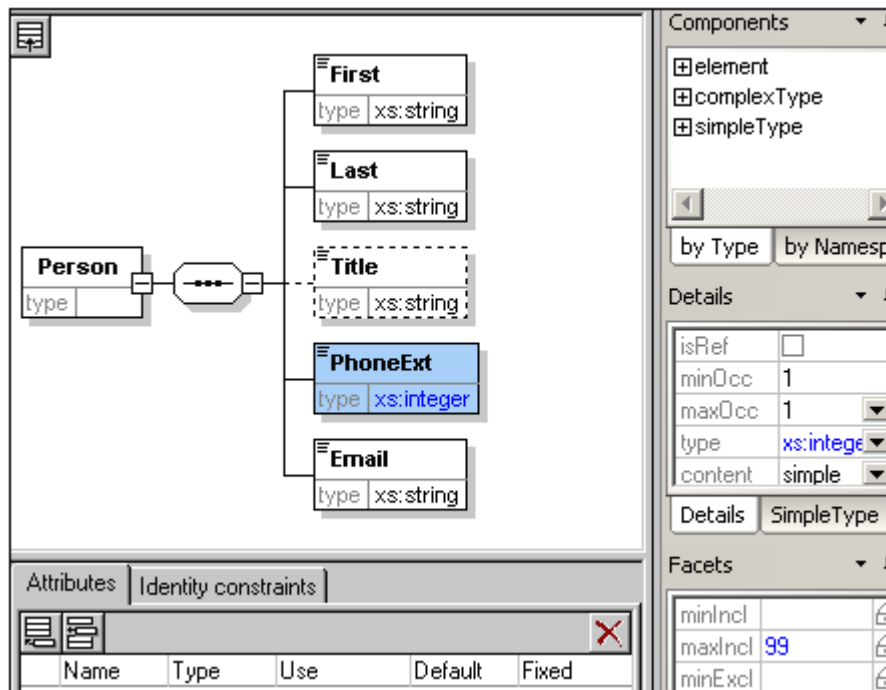
1. Select the menu option **DTD/Schema | Go to definition** or click the Go To Definition

icon . The associated schema, in this case `AddressLast.xsd` is opened. Switch to Schema View (screenshot below).

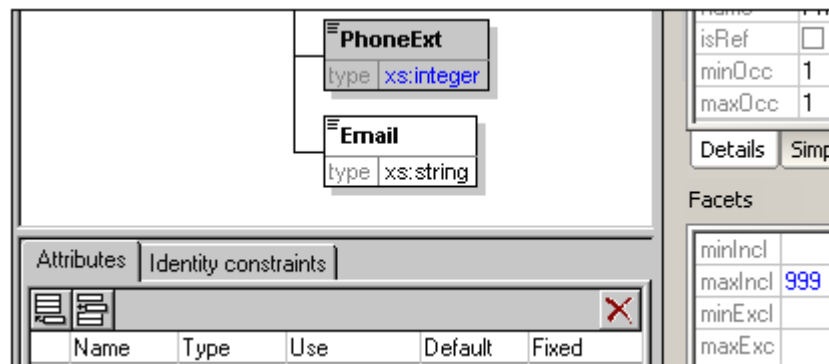


Component	Name	Namespace
element	Company	ann:Root element
complexType	AddressType	ann:
complexType	US-Address	ann:
simpleType	US-State	ann:
complexType	UK-Address	ann:
element	Person	ann:

- Click the Display Diagram icon  of the global `Person` element, and then click the `PhoneExt` element. The facet data in the Facets tab is displayed.



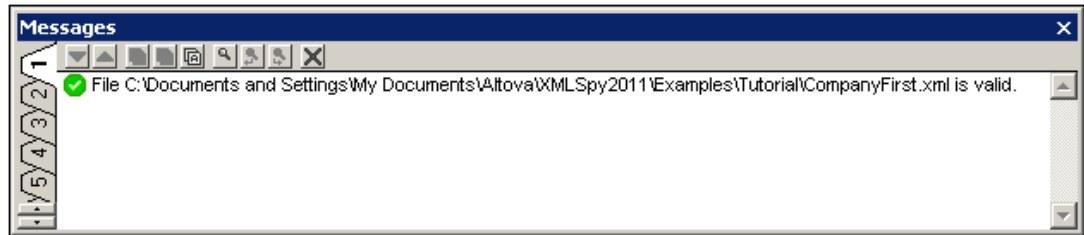
- In the Facets tab, double-click the `maxIncl` value field, change the value 99 to 999, and confirm with **Enter**.



- Save the schema document.

5. Press **Ctrl+Tab** to switch back to the XML document.

6. Click  to revalidate the XML document.



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

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

Please note: The `CompanyLast.xml` file delivered with XMLSpy is in the in the Tutorial folder.

3.6 XSLT Transformations

Objective

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

Method

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

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

Commands used in this section

The following XMLSpy commands are used in this section:



XSL/XQuery | Assign XSL, which assigns an XSL file to the active XML document.



XSL/XQuery | Go to XSL, opens the XSL file referenced by the active XML document.



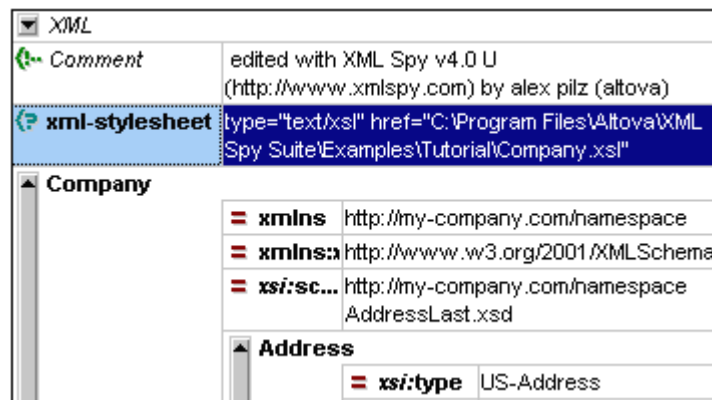
XSL/XQuery | XSL Transformation (F10), or the toolbar icon , transforms the active XML document using the XSL stylesheet assigned to the XML file. If an XSL file has not been assigned then you will be prompted for one when you select this command.

Please note: XMLSpy has two built-in XSLT engines, the Altova XSLT 1.0 Engine and Altova XSLT 2.0 Engine. The Altova XSLT 1.0 Engine is used to process XSLT 1.0 stylesheets. The Altova XSLT 2.0 Engine is used to process XSLT 2.0 stylesheets. The correct engine is automatically selected by XMLSpy on the basis of the version attribute in the `xsl:stylesheet` or `xsl:transform` element. In this tutorial transformation, we use XSLT 1.0 stylesheets. The Altova XSLT 1.0 Engine will automatically be selected for transformations with these stylesheets when the **XSL Transformation** command is invoked.

3.6.1 Assigning an XSLT File

To assign an XSLT file to the `CompanyLast.xml` file:

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

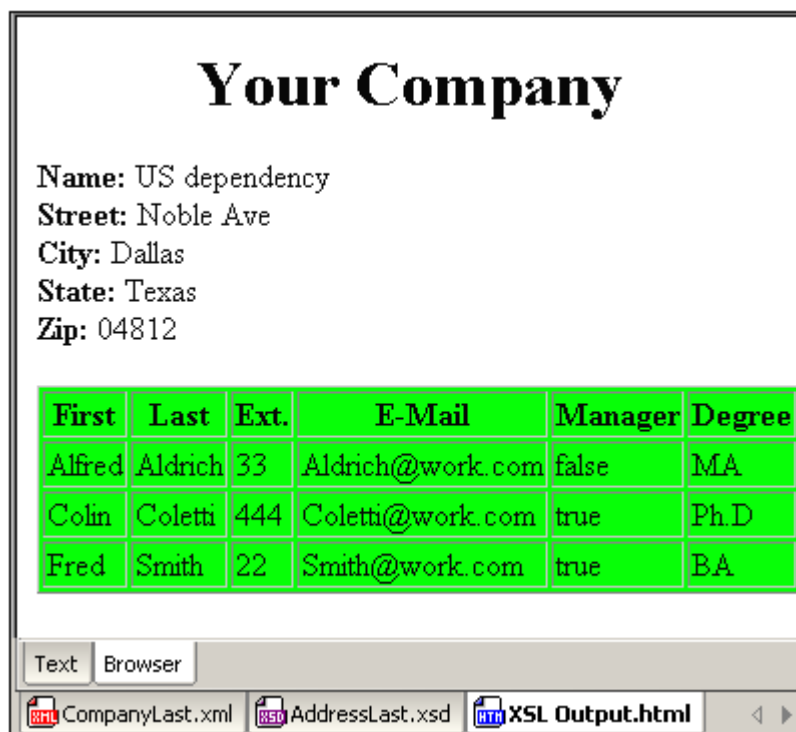


An `xsl:stylesheet` processing instruction is inserted in the XML document that references the XSL file. If you activated the `Make Path Relative to CompanyLast.xml` check box, then the path is relative; otherwise absolute (as in the screenshot above).

3.6.2 Transforming the XML File

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

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



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

3.6.3 Modifying the XSL File

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

Do the following:

1. Click the `CompanyLast.xml` tab to make it the active document, and make sure you are in Grid View.
2. Select the menu option **XSL/XQuery | Go to XSL**.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
  xmlns:my="http://my-company.com/namespace">

  <xsl:template match="/">
    <html>
      <head> <title>Your company</title></head>
      <body>
        <h1><center>Your Company</center></h1>
        <xsl:apply-templates select="//my:Address"/>
        <table border="1" bgcolor="lime">
          <thead align="center">
            <td><strong>First</strong></td>
            <td><strong>Last</strong></td>
            <td><strong>Ext.</strong></td>
```

The command opens the `Company.xsl` file referenced in the XML document.

- Find the line `<table border="1" bgcolor="lime">`, and change the entry `bgcolor="lime"` to `bgcolor="yellow"`.

```
<h1><center>Your Company</center></h1>
<xsl:apply-templates select="//my:Address"/>
<table border="1" bgcolor="yellow">
  <thead align="center">
    <td><strong>First</strong></td>
    <td><strong>Last</strong></td>
```

- Select the menu option **File | Save** to save the changes made to the XSL file.
- Click the `CompanyLast.xml` tab to make the XML file active, and select **XSL/XQuery | XSL Transformation**, or press **F10**. A new `XSL Output.html` file appears in the XMLSpy GUI in Browser View. The background color of the table is yellow.

Your Company					
Name: US dependency					
Street: Noble Ave					
City: Dallas					
State: Texas					
Zip: 04812					
First	Last	Ext.	E-Mail	Manager	Degree
Alfred	Aldrich	33	Aldrich@work.com	false	MA
Colin	Coletti	444	Coletti@work.com	true	Ph.D
Fred	Smith	22	Smith@work.com	true	BA

- Select the menu option **File | Save**, and save the document as `Company.html`.

3.7 Project Management

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

3.7.1 Benefits of Projects

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

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

All the above project settings can be defined using the menu option **Project | Project Properties....** In the next section, you will create a project using the Project menu.

Additionally, the following advanced project features are available:

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

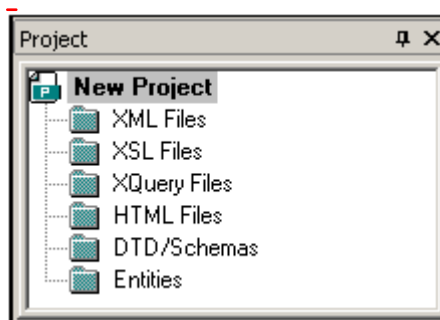
3.7.2 Building a Project

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

Creating a basic project

To create a new project:

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



2. Click the *CompanyLast.xml* tab to make the *CompanyLast.xml* file the active file in the Main Window.
3. Select the menu option **Project | Add active and related files to project**. Two files are added to the project: *CompanyLast.xml* and *AddressLast.xsd*. Note that files

referenced with Processing instructions (such as XSLT files) do not qualify as related files.

4. Select the menu option **Project | Save Project** and save the project under the name `Tutorial`.

Adding files to the project

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

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

Please note: Alternatively, you can right-click a project folder and select Add Active File to add the active file to that specific folder.

Other useful commands

Here are some other commonly used project commands:

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

3.8 That's It

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

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

4 Editing Views

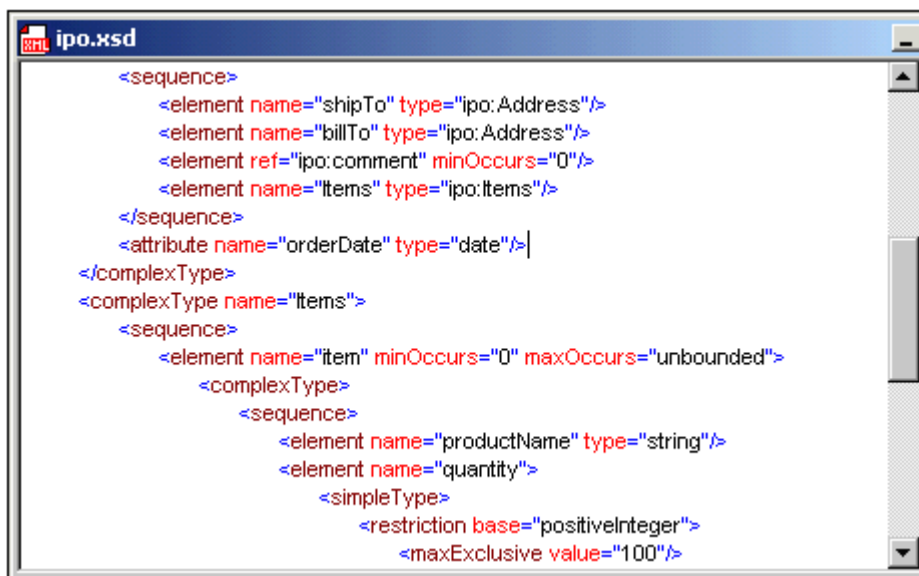
XMLSpy contains powerful editing views. In addition to a Text View with intelligent editing features, there are graphical views that greatly simplify the editing of documents. Depending on what type of document is currently active in XMLSpy, the Main Window will have one or more of XMLSpy's Editing Views.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

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

- [Text View](#)
- [Grid View](#)
- [Schema View](#)
- [WSDL View](#)
- [XBRL View](#)
- [Authentic View](#)
- [Browser View](#)
- [Archive View](#)

4.1 Text View

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



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

The general Text View features have been organized as follows:

- [Formatting in Text View](#) describes how the font properties, indentation, and word-wrapping of the document can be specified.
- [Document display](#) contains information about the line-numbering, bookmarking, expanding/collapsing of nodes, and other display-related features.
- [Editing in Text View](#) describes the features that are available while you edit, particularly the intelligent editing features.
- [Entry helpers](#) are the windows that provide context-sensitive data-entry options. For example, the elements or attributes that can be validly added at a given document location are displayed in an entry helper and any one of these options can be inserted by double-clicking it.

Switching to Text View

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

4.1.1 Formatting in Text View

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

Fonts

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

Text items in a document that have different semantics, can be colored differently. For example, you can color element names, attribute names, and element content differently. When you set different colors for different text items, the syntax-coloring feature is enabled. Text fonts are customized in the [Text Fonts tab of the Options dialog](#), and how to do this is described in the section, [User Reference | Options | Text Fonts](#) section of this documentation.

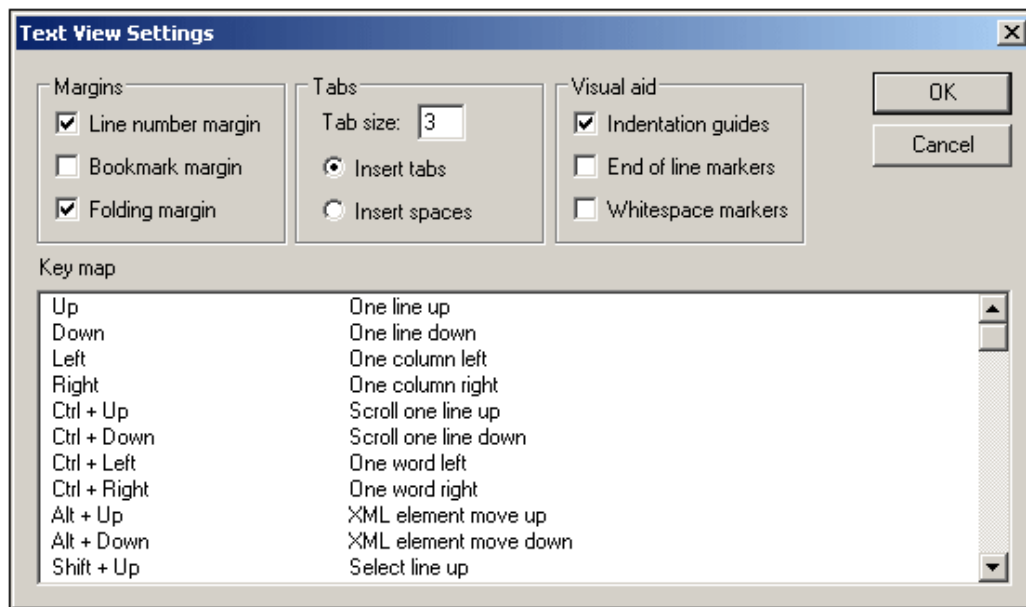
Indentation

Well-formed XML documents can be pretty-printed. This means that the document can be formatted so that the hierarchical structure of the document is displayed using new lines and indentations (*see screenshot below*).



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

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



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

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

Using tabs and spaces for formatting

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

Word-wrapping

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

4.1.2 Displaying the Document

Text View has visual features to make the display and editing of large sections of text easier. Some very useful features are: (i) [Line Numbers](#), (ii) [Bookmarks](#), (iii) [Source Folding](#) (expanding and collapsing the display of nodes), (iv) [Indentation Guides](#), and (v) [End-of-Line and Whitespace Markers](#). These commands are available in the Text View Settings dialog (*first screenshot below*) and the Text toolbar (*second screenshot below*).

The Text View Settings dialog is accessed via the **View | Text View Settings** command or the **Text View Settings** button in the Text toolbar. Settings in the Text View Settings dialog apply to

Source folding

Source folding refers to the ability to expand and collapse nodes (*screenshot below*) and is displayed in the source folding margin. The margin can be toggled on and off in the Text View Settings dialog (see *screenshot above*). In the screenshot below, notice how the line numbering at Lines 14 and 24 has been collapsed together with the collapsed nodes.



The **Toggle All Folds** command in the Text toolbar toggles **all** nodes to their expanded forms or collapses all nodes to the top-level document element. After a node has been expanded, the following options are available when clicking on the node's toggle icon (+/- icon):

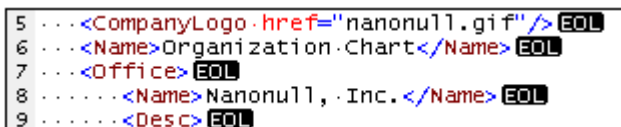
- **Plain click:** Collapses the node. Clicking on it again expands the node so that descendant nodes are shown expanded or collapsed according to how they were before the node was collapsed.
- **Shift+Click:** Collapses all descendant nodes up to the level of the children nodes. The clicked node itself is open and shows the collapsed children nodes. This will work for all collapsible nodes in a branch of the hierarchy except the last collapsible node in that branch.
- **Ctrl+Click:** If a node or any of its descendants is collapsed, **Ctrl+Click** expands all collapsed descendant nodes. In short, this means that all descendant nodes of the control-clicked node are expanded.

Indentation guides

Indentation guides are vertical dotted lines that indicate the extent of a line's indentation (see *screenshot above*). They can be toggled on and off in the Text View Settings dialog.

End-of-line markers, whitespace markers

End-of-line (EOL) markers and whitespace markers can be toggled on in the Text View Settings dialog. The screenshot below shows these markers in the document display; each dot represents a whitespace.



Zooming in and out

You can zoom in and out of Text View by scrolling (with the scroll-wheel of the mouse) while keeping the **Ctrl** key pressed. This enables you to magnify and reduce the size of text in Text View. If you wish to increase the size of fonts, do this in the [Options dialog](#).

Go to line/character

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

4.1.3 Editing in Text View

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

- [Syntax coloring](#)
- [Start-tag and end-tag matching](#)
- [Intelligent editing](#)
- [Auto-completion](#)
- [Moving siblings relative to each other](#)
- [Selecting an entire element](#)
- [Drag-and-drop and context menus](#)
- [Find and replace](#)
- [Unlimited undo](#)
- [Spelling check](#)

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

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

Syntax coloring

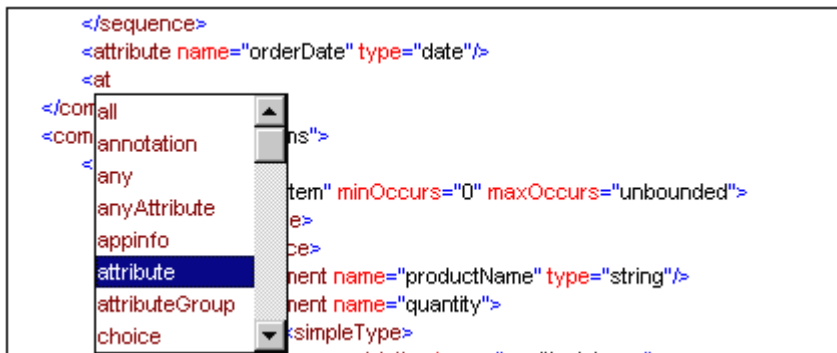
Syntax coloring is applied according to the semantic value of the text. For example, in XML documents, depending on whether the XML node is an element, attribute, content, CDATA section, comment, or processing instruction, the node name (and in some cases the node's content) is colored differently. Four groups of document type are distinguished: (i) generic XML (which includes HTML); (ii) XQuery; (iii) CSS; and (iv) JSON. The text properties (including color) of each group can be set in the Text Fonts tab of the Options dialog (**Tools | Options**).

Start-tag and end-tag matching

When you place the cursor inside a start or end tag of a markup element, pressing **Ctrl+E** highlights the other member of the pair. Pressing **Ctrl+E** repeatedly enables you to switch between the start and end tags. This is an excellent aid to locating the start and end tags of an XML element.

Intelligent Editing

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



Pop-up windows also appears in the following cases:

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

Auto-completion

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

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


```

<Person>
  <First
    First (given) name of person
  </First>
  <PhoneEXT>
  </PhoneEXT>
  <EMail>
  </EMail>
  <LeaveTotal>
  </LeaveTotal>
  <LeaveUsed>
  </LeaveUsed>
  <LeaveLeft>
  </LeaveLeft>
</Person>

```

Auto-completion can be switched on and off in the [Editing tab of the Options dialog](#) (**Tools | Options | Editing**).

Moving sibling elements relative to each other

When the cursor is within an element, pressing **Alt+ArrowUp** or **Alt+ArrowDown** moves the selected element up or down relative to its siblings.

Selecting an entire element

When the cursor is within an element, pressing **Ctrl+Shift+E** selects the entire element.

Drag-and-Drop and Context Menus

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

Find and Replace

You can use the [Find](#) and [Replace](#) commands to quickly locate and change text. These commands also take regular expressions as input, thereby giving you powerful search capabilities. (See [Edit | Find](#) for details.)

Unlimited Undo

XMLSpy offers unlimited levels of [Undo](#) and [Redo](#) for all editing operations.

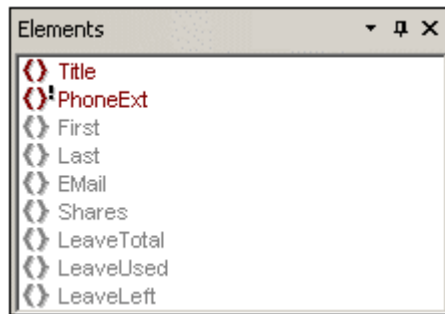
Spelling check

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

4.1.4 Entry Helpers in Text View

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

- **XML:** Elements (*screenshot below*), Attributes, Entities



- *HTML*: Elements, Attributes, Entities
- *CSS*: CSS Outline, CSS Properties, HTML Elements
- *DTD*: None
- *XQuery*: XQuery Keywords, XQuery Variables, XQuery Functions
- *WSDL*: Overview, Details
- *Text*: Entities

Note that several document types, such as XSD, XSLT, XHTML, and RDF, are essentially XML documents and will therefore have the Elements, Attributes, and Entities entry helpers.

Display and use of entry helper items

Different items in the various entry helpers are variously color-coded. These color codes are explained in the Entry Helpers documentation of the respective document types. In general, the following points should be noted about entry helpers:

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

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

4.2 Grid View

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

XML

Company

=	xmlns	http://my-company.com/namespace
=	xmlns:xsi	http://www.w3.org/2001/XMLSchema-instance
=	xsi:schema...	http://my-company.com/namespace AddressLast.xsd

Address

=	xsi:type	US-Address
⌂	Name	US dependency
⌂	Street	Noble Ave.
⌂	City	Dallas
⌂	Zip	04812
⌂	State	Texas

Person (3)

	=	Manager	=	Degree	=	Programmer	⌂	First	⌂	Last
1	false		MA		true			Alfred		Aldrich
2	true		Ph.D		false			Colin		Coleman
3	true		BA		false			Fred		Smith

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

Display and navigation

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

Customizing Grid View

- To resize columns, place the cursor over the appropriate border and drag so as to

- achieve the desired width.
- To resize a column to the width of its largest entry, double-click on the grid line to the right of that column.
- To adjust column widths to display all content, select the menu item [View | Optimal widths](#) command, or click on the Optimal widths icon .
- The heights of cells are determined by their contents. They can be adjusted with the menu option **Tools | Options | View | Grid View**, "Limit cell height to xx lines".

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

Intelligent editing

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

Grid View tables

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

4.2.1 Editing in Grid View

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

Inserting or appending element or attribute names

To insert or append an element or attribute relative to a selected item, you can use either commands in the XML menu or icons in the **Attributes and Elements Toolbar**



If this toolbar is not visible, select the menu option **Tools | Customize**, and, in the **Toolbars** tab, check the **Attr & Elem** entry. For a detailed explanation of how to use the toolbar icon commands and XML menu commands, see the [XML Menu](#) section.

To insert or append an element or attribute:

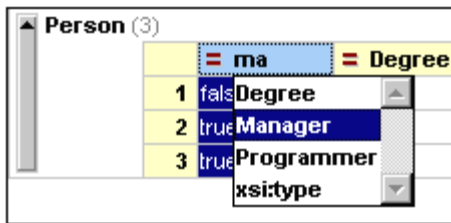
1. Select the item relative to which the element or attribute is to be inserted or appended.
2. Click on the appropriate icon in the Attributes and Elements toolbar, or select the appropriate command from the [XML menu](#) (**Insert**, **Append**, or **Add Child**). This creates a new entry in the grid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

You will notice that the various [Entry Helpers](#) are constantly updated depending on the current selection in the Grid View. The [Info Window](#) constantly shows important information regarding the selected element or attribute.

Editing an element or attribute name

When you edit the name of an element or attribute, a popup menu containing the available options opens. These options depend on the position of the element and the content model

defined by the schema. A similar popup is displayed if the contents of an element or attribute are restricted by an enumeration or choice of some sort.



To edit an element or attribute name:

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

Adding namespace prefixes

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

Grid View context menu

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

Spelling check

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

4.2.2 Grid View Tables

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

For instance, consider the following XML document:

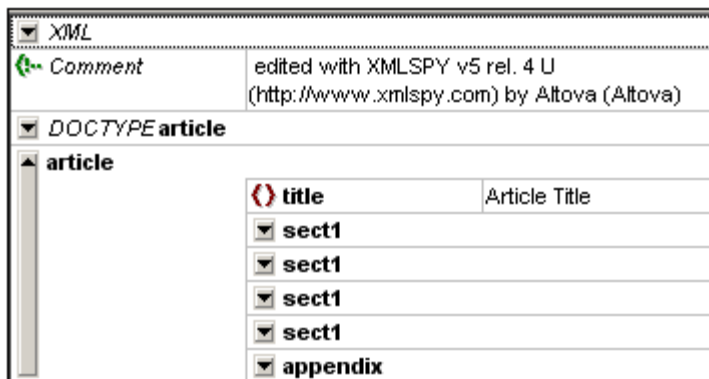
```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <!-- edited with XMLSPY v5 rel. 4 U (ht
3  <!DOCTYPE article SYSTEM "C:\Program Fi
4  <article>
5      <title>Article Title</title>
6      <sect1>
7          <title>Section 1</title>
8          <simpara/>
9      </sect1>
10     <sect1>
11         <title>Section 2</title>
12         <para/>
13     </sect1>
14     <sect1>
15         <title>Section 3</title>
16         <para/>
17         <para/>
18     </sect1>
19     <sect1>
20         <title>Section 4</title>
21         <para/>
22     </sect1>
23     <appendix>
24         <title/>
25         <para/>
26     </appendix>
27 </article>

```

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

The normal Grid View of this document is as follows:



Now here is the Table View of this document—more correctly, that of the `article` element. (To get this view: select the `article` element in normal Grid View (by clicking on it) and click the **Display as Table** icon . Alternatively, select the menu item **XML | Table | Display as Table (F12)**.)

	title	sect1	appendix
1	Article Title	sect1	appendix
		sect1	
		sect1	
		sect1	

Notice that each child element of `article` (the element displayed as a table)—that is, `title`, `sect1`, and `appendix`—has been assigned a column and that each occurrence of each child-type is listed in the appropriate column. Note also that the table structure extends only to the child level (the `sect1` elements themselves are not displayed as a table). In order to display `sect1` elements as a table, select any of the `sect1` elements in the `sect1` column, and click . The `sect1` elements are now displayed in a table (see screenshot below): each of their child elements is assigned a column in the `sect1` table.

	title	simpara	para
1	Section 1		
2	Section 2		para (1)
3	Section 3		para (2)
4	Section 4		para (1)

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

Please note:

- Attributes are also considered child nodes, and columns are also created for attributes.
- You can switch between normal Grid View and Table View by selecting the desired element and clicking or **F12**.
- If you are viewing the document in Table View and you switch to Text View, when you switch back to Grid View the document will display in normal Grid View.

Manipulating table data

You can manipulate table data in the following ways:

- Drag-and-drop column headers to move columns.
- Sort column data for text nodes using the menu command [XML | Table | Ascending Sort](#) (also **Descending Sort**).
- Append (or insert) rows using the menu command [XML | Table | Insert Row](#) (also **Append Row**).

Moving data between Table View and external applications

You can take advantage of the table structure of data in Table View to exchange data between

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

	pic_filetype	book_isbn	book_title
1	jpg	0764547097	XHTML: Migrating Applications Toward
2	jpg	0471375225	XML Database Toolkit: Resources & Development
3	jpg	1861003110	Professional XML
4	gif	007212220X	XML Elements of Style
5	gif	0782122663	Mastering XML
6	gif	0764532367	XML Bible
7	jpg	1565925807	DocBook: The Definitive Guide
8	jpg	0130147141	The XML Handbook - 2nd Edition
9	jpg	0123555922	XML for Real Programmers
10	jpg	0201616335	The Unicode Standard, Version 3.0
11	jpg	1565922247	CJKV Information Processing

The screenshot above shows 11 book elements displayed as a table in Table View. To copy the entire table into an Excel sheet, you would select the `book ( 11 )` element, copy it as structured text, and paste it into an Excel sheet. If you were to select cell A1 and paste, the result would be as shown below.

	A	B	C
1	pic_filetype	book_isbn	book_title
2	jpg	764547097	XHTML: Migrating Applications Toward XML
3	jpg	471375225	XML Database Toolkit: Resources & Techniques for Improved
4	jpg	1861003110	Professional XML
5	gif	007212220X	XML Elements of Style
6	gif	782122663	Mastering XML
7	gif	764532367	XML Bible
8	jpg	1565925807	DocBook: The Definitive Guide
9	jpg	130147141	The XML Handbook - 2nd Edition
10	jpg	123555922	XML for Real Programmers
11	jpg	201616335	The Unicode Standard, Version 3.0
12	jpg	1565922247	CJKV Information Processing

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

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

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

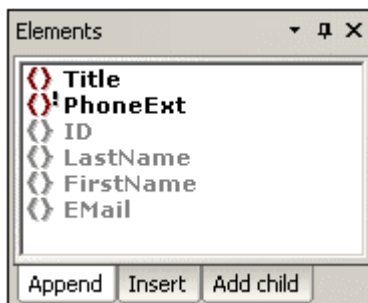
- If data already exists in these cells in Table View, the new data overwrites the original data.
- If more rows are required to accommodate the new data, these are created.
- If more columns are required to accommodate the new data, these are **not** created.
- The data in the cells becomes the contents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

For more complex data exchange tasks, XMLSpy also offers a set of unique conversion functions that let you directly [import or export XML data](#) from any text file, Word document or database.

4.2.3 Entry Helpers in Grid View

Elements Entry Helper

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



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

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

Attributes Entry Helper

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



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

Please note: Existing attributes, which cannot legally be added to the current element a second time, are shown in gray.

Entities Entry Helper

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

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

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

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

4.3 Schema View

XML Schemas can be viewed and edited in Schema View. Schema View itself has two views:

- **Schema Overview** displays all global components, such as global elements and complex types, in a simple tabular list ([see screenshot](#))
- **Content Model View** shows the content models of individual global components ([see screenshot](#))

You can switch from Schema Overview to the content model of a specific global component by clicking the icon of that global component. To return to Schema Overview, click the Show

Globals icon  in Content Model View.

Organization of this section

This section has been organized into the following sub-sections

- [Schema Overview](#) describes how global components are displayed in Schema Overview and how they are edited
- [Content Model View](#) describes how the content models of individual global components are edited
- [Entry Helpers](#) explains the organization of the entry helpers in Schema View
- [Identity Constraints](#) describes how identity constraints are displayed and edited in Schema View
- [Smart Restrictions](#) is a XMLSpy editing feature that facilitates the graphical creation and editing of derived types from their base types; this section describes how the Smart Restrictions feature is used
- A description of how the [xml: prefixed attributes](#) `base`, `id`, `lang`, and `space` can be added graphically to schema components
- [Back and Forward: Moving through Positions](#) explains a Schema View feature that enables you to move through previously viewed positions

Connecting to SchemaAgent

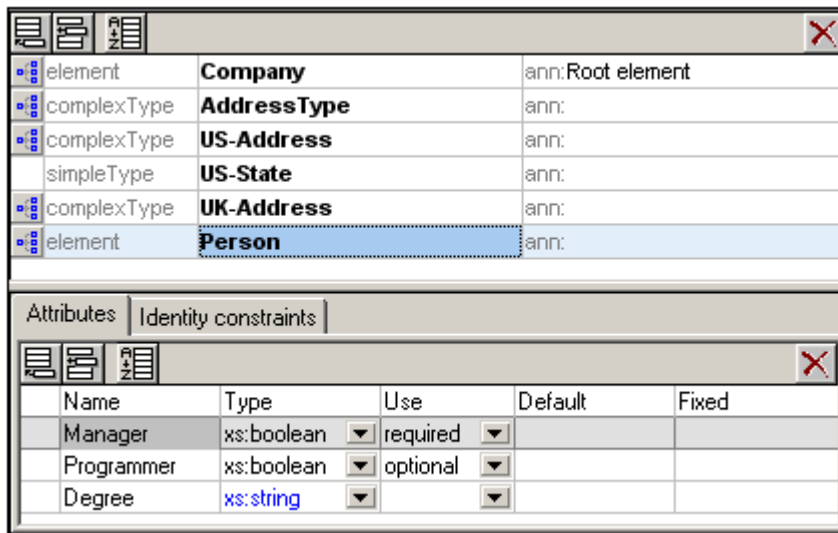
From XMLSpy you can also connect to SchemaAgent in order to display components from other schemas in the GUI and to use these components in the schema being currently edited. How to work with SchemaAgent in XMLSpy is described in the section [Working with SchemaAgent](#).

Find in schemas

The Find in Schemas features enables intelligent searches in schemas, i.e., searches that are restricted by various schema-related criteria. For example, searches may be restricted to certain component types, thus making the search more efficient. Find in Schemas is described in the [DTDs and XML Schemas section](#).

4.3.1 Schema Overview

Schema Overview displays a list of all the global components of the schema (global elements, complex types, etc).



You can insert, append, or delete global components, as well as modify their properties. To insert, append, or delete, use the respective buttons at the top of Schema Overview. To modify properties, select the required component in the Schema Overview list, and edit its properties in either the entry helpers (at right of view) or the Attributes/Identity Constraints pane (at bottom of view).

You can edit the content model of a global component in [Content Model View](#), which is accessed by clicking the Content Model View icon  at the left of the global component.

Note the following editing features of Schema Overview:

- You can reposition components in the Schema Overview list using drag-and-drop.
- You can navigate using the arrow keys of your keyboard.
- You can copy or move global components, attributes, and identity constraints to a different position and from one schema to another using cut/copy-and-paste.
- Right-clicking a component opens a context menu that allows you to cut, copy, paste, delete, or edit the annotation data of that component.

Global components in Schema Overview

At the top level of an XML Schema document (i.e., at the level of children of the `schema` element), the following five basic components can be defined:

- Annotation
- Type definition (simple or complex)
- Declaration (element or attribute)
- Attribute group
- Model group

These components are called **global components**. The **Schema Overview** displays a list of all global components in your schema in a tabular form. Some global components (such as complex types, element declarations, and model groups) can have a content model which describes the component's structure and contents. Other global components (such as annotations, simple types, and attribute groups) do not have a content model. Those components for which content models are possible have a  icon to the left of the component name. Clicking on this icon opens the [Content Model View](#) for that global component.

Key terms

- *Simple type and complex type.* A simple type is used to define all attributes and elements that contain only text and that have no associated attribute. A simple type, therefore, has no content model—only text (which can be restricted by the datatype). A complex type is one that has at least one child element or attribute. Declaring a child element on an element automatically assigns the element a type of complex.
- *Global and local components.* A global component can be any of the five listed above. A global component can be defined in Schema Overview, and it then immediately appears in the list of global components in Schema Overview. If the global component is a complex type, an element declaration, or a model group, you can subsequently define its content model by editing it in Content Model View. Once a global component has been defined, it can be referenced by local components. A local component is created directly within the content model of some component. Note that, in the Content Model View, a local component can be converted into a global component (via the right-click context menu).

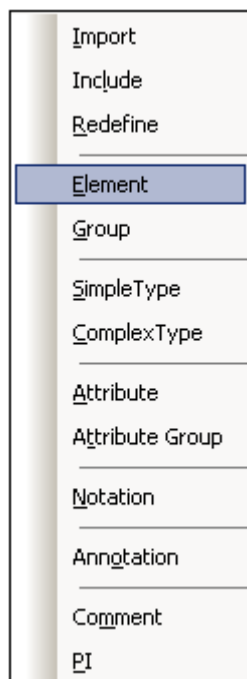
Comments and processing instructions

In XML Schema documents, comments and processing instructions within simple types and complex types are collected and moved to the end of the enclosing object. In such cases, it is therefore recommended that you use annotations instead of comments.

Creating global components

To create a global component in Schema Overview:

1. Click the Insert  or Append  icon at the top of the Schema Overview. This pops up a menu listing the various component types (element, simple type, complex type, model group, etc).



2. Select the type of component you want. An entry of that type is created in the list of global components.
3. Enter the name of the component in the entry, and press **Enter**. The name of the new

global component is added to the appropriate list/s (Elm, Grp, Com, Sim, etc.) in the Component Navigator entry helper. You can edit the content model of the new global component either by double-clicking the component name in the Component Navigator or by clicking the  icon to the left of the new component's name in the list of global components.

Please note:

- You can also create a global component while editing in Content Model View. Right-click anywhere in the window and select **New Global | Element**.
- While editing in Content Model View, you can make a local element a global element—or even a complex type if the element has an element or attribute child. Select the local element, right-click anywhere in the window, and select **Make Global | Element** or **Make Global | Complex type**.

Deleting global components

To delete a global component:

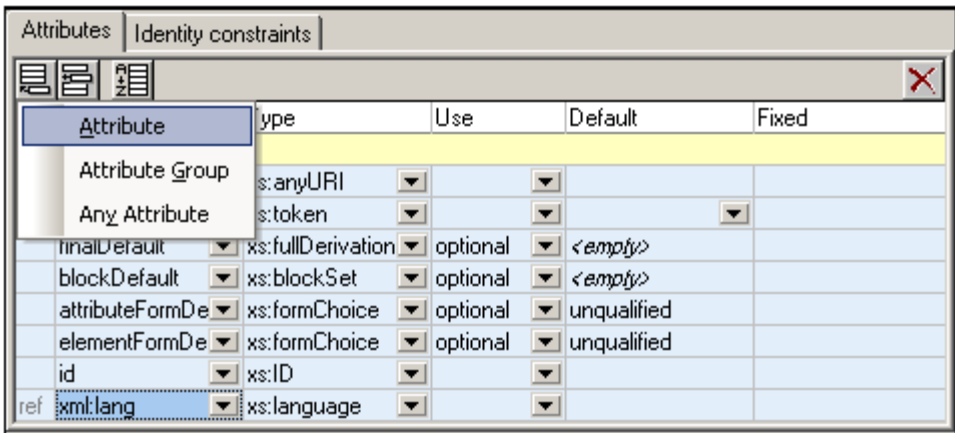
1. Select the global component in the list of global components in the Schema Overview.
2. Press the **Delete** key, or click the **Delete** icon  at the top of the Schema Overview.

Attributes and identity constraints of components

You can define attributes and identity constraints for components in either Schema Overview or Content Model View. In Schema Overview, the attributes and identity constraints of a component are displayed in the Attributes/Identity Constraints pane at the bottom of the Schema Overview window and can be edited there. In Content Model View, attributes and identity constraints can appear either in the Attributes/Identity Constraints pane at the bottom of the Content Model Window or within the Content Model diagram itself, where it can be edited. You can select how to display attributes and identity constraints in Content Model View with a setting in the [Schema Display Configuration dialog](#), which is accessed with the **Schema design | Configure view** menu command.

Defining attributes for a component

To define attributes for a component, you use the Attributes/Identity Constraints pane, which is at the bottom of the Schema Overview window.



Attribute	type	Use	Default	Fixed
Any Attribute	xs:anyURI			
Any Attribute	xs:token			
finalDefault	xs:fullDerivation	optional	<empty>	
blockDefault	xs:blockSet	optional	<empty>	
attributeFormDe	xs:formChoice	optional	unqualified	
elementFormDe	xs:formChoice	optional	unqualified	
id	xs:ID			
ref	xs:language			

To define attributes for a global component for which attributes are allowed:

1. Select the global component in the global components list.

2. In the Attributes/Identity Constraints pane, select the Attributes tab.
3. Click the Append  or Insert  icon at the top left of the Attribute tab.
4. From the popup that appears, select the attribute type you want to append or insert. An entry is created in the Attribute list.
5. In the newly created entry, enter the attribute's properties.

Please note: You can also define attributes for global components in Content Model View: Select the global component, and then define attributes as described above.

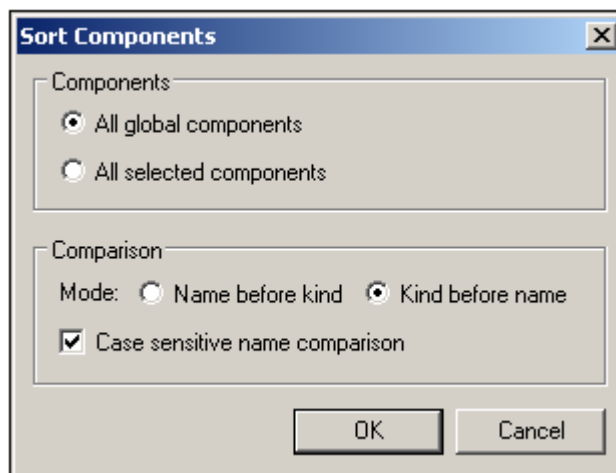
Defining identity constraints for a component

To define identity constraints for a component, you use the Attributes/Identity Constraints pane, which is at the bottom of the Schema Overview window. See [Identity Constraints](#) for a description of how to work with identity constraints.

Sorting global components, attributes, and identity constraints

Global components can be sorted alphabetically, in ascending order, by local name and kind. Do this as follows:

1. If you wish to sort a selection of global components, then select the global components you wish to sort by clicking them with the **Ctrl** key pressed. (You can also select a range by pressing the **Shift** key and clicking the first and last components of the range.) The global components you select will be highlighted. If you wish to sort all the global components you can select one or more global components. An option for sorting all the components will be available.
2. Clicking the **Sort** icon  at the top of the Schema Overview pane. This pops up the Sort Components dialog (screenshot below).



3. In the *Components* pane, select whether all the components or only the selection is to be sorted. The *Selected Components* option is enabled only if a the number of selected global components is greater than one.
4. In the *Comparison* pane, you can select whether the sort is performed first on the criterion of local name or component kind. If there is more than one component of the same local name or same component kind (whichever is the first sort criterion), then these are sorted on the secondary criterion.
5. For case-sensitive name comparison, select the check box with this label. If this option is selected, lowercase will be listed before uppercase.

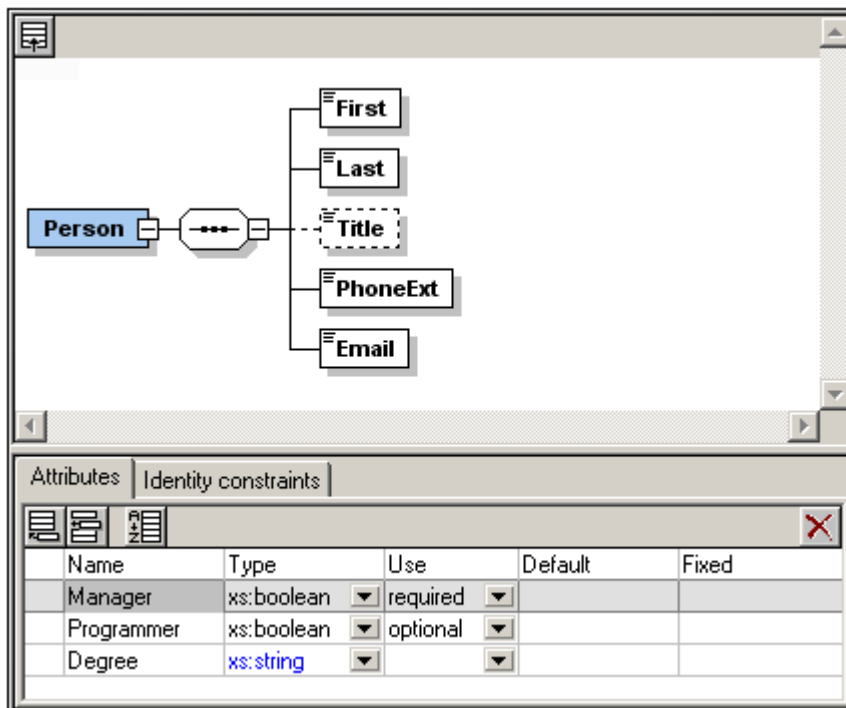
Sorting attributes and identity constraints works in the same way as sorting global

components. The **Sort** icon  is located in the toolbar of the Attributes/Identity Constraints pane, and it is enabled if the pane contains at least two named sibling components (such as attributes or keys).

4.3.2 Content Model View

A content model is a description of the structure and contents of an element. Global components which can have a content model (for example, elements, complex types, and model groups; but not, for example, simple types) are indicated in the Schema Overview list with a  icon to the left of the component name. Clicking on this icon opens the Content Model View for that global component. Alternatively, (i) select a component and then select the menu option **Schema design | Display Diagram**, or (ii) double-click on a component's name in the [Component Navigator](#) (which is the entry helper located, by default, at top right). Note that only one content model in the schema can be open at a time. When a content model is open, you can jump to the content model of a component within the current content model by holding down **Ctrl** and double-clicking the required component.

The content model is displayed in the Content Model View as a tree (see screenshot below). You can configure the appearance of the tree in the Schema Display Configuration dialog (menu item [Schema design | Configure view](#)).



Note the following editing features of Content Model View:

- Each level (of elements or element groups) in the tree is joined to adjacent levels with a compositor.
- Drag-and-drop functionality enables you to move tree objects (compositors, elements, element groups) around.
- You can use keyboard shortcuts to copy (**Ctrl-c**) and paste (**Ctrl-v**) tree objects
- You can add objects (compositors, elements, and element groups) via the context menu (right-click an object).

- You can sort sibling components by selecting the sibling components you wish to sort, right-clicking, and selecting the **Sort Components** command from the context menu. You can prioritize two sort criteria: (i) local name, and (ii) component kind.
- You can edit the properties of an object in the Details entry helper (compositors, elements, element groups) and the Attributes/Identity Constraints pane.
- The Attributes and Identity Constraints of a component are displayed in a pane at the bottom of the Main Window. Attributes and Identity Constraints can also be displayed in the Content Model diagram instead of in the Attributes/Identity Constraints pane. This viewing option can be set in the [Schema Display Configuration dialog](#).

These features are explained in detail in the subsections of this section and in the tutorial.

To return to the Schema Overview, click the **Show Globals** icon  or select the menu option **Schema design | Display All Globals**.

Editing in Content Model View

Content Model View enables you to quickly define the content model of the following three component types graphically with a few mouse clicks:

- Complex types
- Element declarations
- Model groups

All other schema components (annotations, attribute declarations, simple types, etc.) do not have a content model.

In Content Model View, the various parts of the content model are represented graphically. These parts are organized into two broad groups: **compositors** and **components**. Typically a compositor is added and then the desired child components.

Compositors

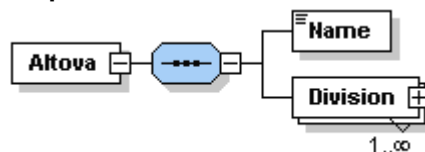
A **compositor** defines the order in which child elements occur. There are three compositors: *sequence*, *choice*, and *all*.

To insert a compositor:

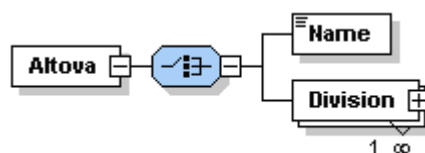
1. Right-click the element to which you wish to add child elements
2. Select **Add Child | Sequence** (or **Choice** or **All**).

The compositor is added, and will look as below:

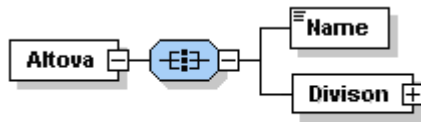
- **Sequence**



- **Choice**



- **All**



To change the compositor, right-click the compositor and select **Change Model | Sequence** (or **Choice** or **All**). After you have added the compositor, you will need to add child element/s or a model group.

Components in the Content Model

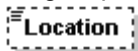
Given below is a list of components that are used in content models. The graphical representation of each provides detailed information about the component's type and structural properties.

- Mandatory single element



Details: The rectangle indicates an element and the solid border indicates that the element is required. The absence of a number range indicates a single element (i.e. `minOcc=1` and `maxOcc=1`). The name of the element is `Country`. The blue color indicates that the element is currently selected; (a component is selected by clicking it). When a component is not selected, it is white.

- Single optional element



Details: The rectangle indicates an element and the dashed border means the element is optional. The absence of a number range indicates a single element (i.e. `minOcc=0` and `maxOcc=1`). Element name is `Location`.

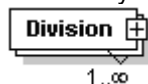
Note: The context menu option **Optional** converts a mandatory element into an optional one.

- Mandatory multiple element



Details: The rectangle indicates an element and the solid border indicates that the element is required. The number range `1..5` means that `minOcc=1` and `maxOcc=5`. Element name is `Alias`.

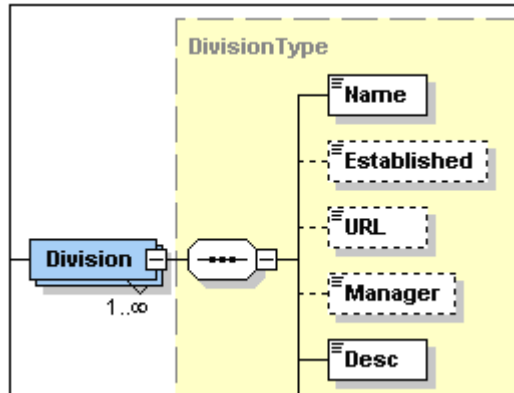
- Mandatory multiple element containing child elements



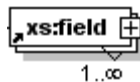
Details: The rectangle indicates an element and the solid border indicates that the element is required. The number range `1..infinity` means that `minOcc=1` and `maxOcc=unbounded`. The plus sign means complex content (i.e. at least one element).

or attribute child). Element name is `Division`.

Note: The context menu option **Unbounded** changes `maxOcc` to `unbounded`. Clicking on the + sign of the element expands the tree view and shows the child elements.



- Element referencing global element



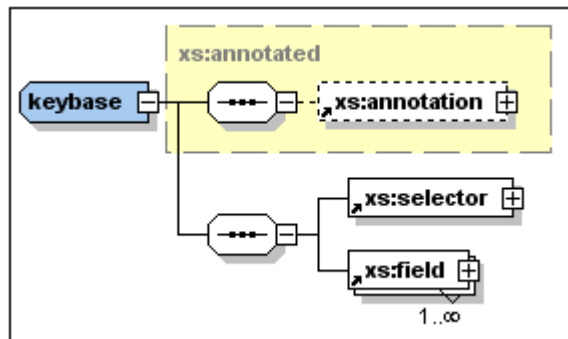
Details: The arrow in the bottom-left means the element references a global element. The rectangle indicates an element and the solid border indicates that the element is required. The number range `1..infinity` means that `minOcc=1` and `maxOcc=unbounded`. The plus sign indicates complex content (i.e. at least one element or attribute child). Element name is `xs:field`.

Note: A global element can be referenced from within simple and complex type definitions, thus enabling you to re-use a global declaration at multiple locations in your schema. You can create a reference to a global element in two ways: (i) by entering a name for the local element that is the same as that of the global element; and (ii) by right-clicking the local element and selecting the option **Reference** from the context menu. You can view the definition of a global element by holding down **Ctrl** and double-clicking the element. Alternatively, right-click, and select **Go to Definition**. If you create a reference to an element that does not exist, the element name appears in red as a warning that there is no definition to refer to.

- Complex type



Details: The irregular hexagon with a plus sign indicates a complex type. The complex type shown here has the name `keybase`. This symbol indicates a global complex type. A global complex type is declared in the Schema Overview, and its content model is typically defined in Content Model View. A global complex type can be used either as (i) the data type of an element, or (ii) the base type of another complex type by assigning it to the element or complex type, respectively, in the Details entry helper (in either Content Model View or in Schema Overview).

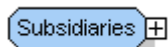


The `keybase` complex type shown above was declared in Schema Overview with a base type of `xs:annotated`. The base type is displayed as a rectangle with a dashed gray border and a yellow background color. Then, in Content Model View, the child elements `xs:selector` and `xs:field` were created. (Note the tiny arrows in the bottom left corner of the `xs:selector` and `xs:field` rectangles. These indicate that both elements reference global elements of those names.)

A local complex type is defined directly in Content Model View by creating a child element or attribute for an element. There is no separate symbol for local complex types.

Please note: The base type of a content model is displayed as a rectangle with a dashed gray border and a yellow background color. You can go to the content model of the base type by double-clicking its name.

- Model group



Details: The irregular octagon with a plus sign indicates a model group. A model group allows you to define and reuse element declarations.

Note: When the model group is declared (in Schema Overview) it is given a name. You subsequently define its content model (in Content Model View) by assigning it a child compositor that contains the element declarations. When the model group is used, it is inserted as a child, or inserted or appended within the content model of some other component (in Content Model View).

- Wildcards



Details: The irregular octagon with `any` at left indicates a wildcard.

Note: Wildcards are used as placeholders to allow elements not specified in the schema or from other namespaces. `##other` = elements can belong to any namespace other than the target namespace defined in the schema; `##any` = elements can belong to any namespace; `##targetNamespace` = elements must belong to the target namespace defined in the schema; `##local` = elements cannot belong to any namespace; `anyURI` = elements belong to the namespace you specify.

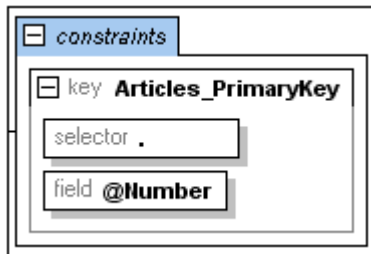
- Attributes



Details: Indicated with the word '*attributes*' in italics in a rectangle that can be expanded. Each attribute is shown in a rectangle with a (i) dashed border if the attribute is options, or (ii) a solid border if the attribute is required (mandatory).

Note: Attributes can be edited in the Details Entry Helper. Attributes can be displayed in the Content Model View diagram or in a pane below the Content Model View. You can toggle between these two views by clicking the Display Attributes  icon. When attributes are displayed in the Content Model View diagram, all attributes of a single element are displayed in a rectangle with a dashed border. To change the order of attributes of an element, drag the attribute outside the containing box and drop when the arrow appears at the required location.

- Identity constraints



Details: Indicated with the word '*constraints*' in italics in a rectangle that can be expanded.

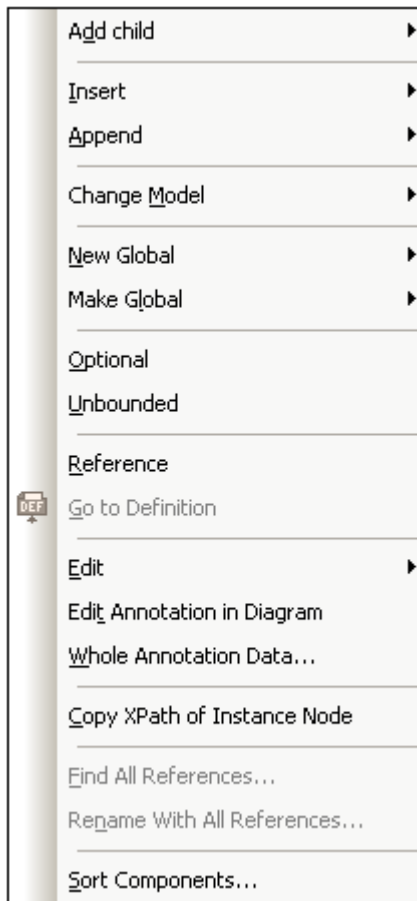
Note: The identity constraints listed in the content model of a component show constraints as defined with the `key` and `keyref` elements, and with the `unique` element. Identity constraints defined using the `ID` datatype are not shown in the content model diagram, but in the Details Entry Helper. Identity constraints can be displayed and edited in the Content Model View or in the Identity Constraints tab of Schema Overview. In Content Model View, you can toggle the Constraints box on and off with the Display Constraints  icon..

Please note:

- Property descriptor lines you have defined in the [Schema Display Configuration dialog](#) can be turned on and off by clicking the Add Predefined Details  toolbar icon.
- You can toggle Attributes and Identity Constraints to appear either in the diagram of the content model itself or in a pane below the Content Model View by clicking the Display in Diagram icons for attributes and constraints,  and  respectively.
- In Content Model View, you can jump to the content model view of any global component within the current content model by holding down the **CTRL** key and double-clicking the required component. You can go to the content model of a base type by double-clicking the name of the base type.

Other editing operations in Content Model View

Editing operations in Content Model View are carried out via the context menu (*see screenshot*) that appears when you right-click within Content Model View. A description of the operations are given below.



Adding child compositors/components and inserting/appending compositors/components

1. Right-click the compositor or component. This opens the context menu (with only the allowed operations enabled).
2. Select the required operation from the context menu.

Changing a compositor

1. Right-click the compositor you want to change.
2. Select the context menu option **Change Model** and, from the sub-menu, select the compositor to which you want to change. (The currently selected compositor is checked.) If a compositor is not allowed at that point, it is disabled.

Creating global components

- To create a new global component, right-click anywhere in Content Model View, select **New Global**, and, from the sub-menu, the required component.
- To make a local element a global element or global complex type, right-click the local element, select **Make Global**, and, from the sub-menu, select either **Element** or **Complex type**. If any of these components cannot legally be created, then it is disabled.

Changing the occurrence definition

You can toggle the minimum and maximum occurrences values of a compositor between 0 and 1 (for `minOccurs`) and 1 and unbounded (for `maxOccurs`), respectively.

Do this as follows:

1. Right-click the compositor or component for which the occurrence value has to be changed.
2. Select the context menu option **Optional** to set `minOccurs` to 0, deselect **Optional** to set `minOccurs` to 1. Select the context menu option **Unbounded** to set `maxOccurs` to unbounded, deselect **Unbounded** to set `maxOccurs` to 1. A check mark appears to the left of the respective menu item when that menu item is selected.

Toggling between local definition and global definition

If a global element exists that has the same name as a local element, then you can toggle between referencing the global definition and using the local definition.

Do this as follows:

1. Right-click the element.
2. Select the context menu option **Reference**. If the global element is referenced, then the menu item is checked. If the local definition is used, the **Reference** item in the menu is not checked.

Jumping to another definition

When you are within a content model, you can jump to the definition of any global component that is contained in that content model.

Do this as follows:

1. Right-click the global component. The global component could be the yellow rectangle of a base type; an element that references a global element; or a model group.
2. Select the context menu option **Go to Definition**. This opens the Content Model View of that global component.

Alternatively, double-click the name of the base type, or press **Ctrl** and double-click the referencing element or the model group.

Editing element names and types

1. Right-click the element.
2. Select the context menu option **Edit | Name** and edit the name, or select **Edit | Type** and edit the type.

Alternatively, double-click the element name, and type in the change.

Creating and editing documentation for a compositor or component

You can add documentation to individual compositors and components as a guide for schema editors.

Do this as follows:

1. Right-click the compositor or component.



2. Select the context menu option **Edit Annotation**. This highlights the documentation space below the compositor/component, in which you can enter descriptive text about the compositor or component. In Text View, the `annotation` and `annotation/documentation` elements will have been created and the `documentation` element will contain the descriptive text you enter.

Alternatively, you can right-click the compositor or component and select Whole Annotation Data. In the Annotation dialog that opens, you can append or insert a documentation item and enter content for it.

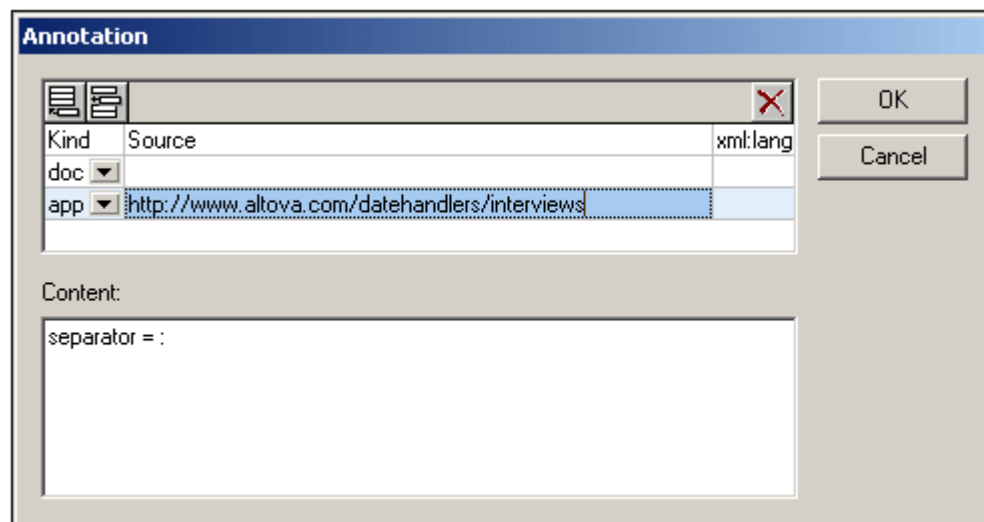
In order to edit pre-existing documentation text, you can use either of the two methods described above, but a quicker method is to double-click the annotation in the diagram and edit directly.

Creating and editing application info for a compositor or component

1. Right-click the compositor or component.



2. Select the context menu option **Whole Annotation Data**. The Annotation dialog box opens (see screenshot below). If annotation (either documentation or appinfo) exists for that element, then this is indicated by a corresponding row in the dialog.



3. To create an `appinfo` element, click the Append  or Insert  icon at top left to append or insert a new row, respectively.
4. In the Kind field of the new row, select the `app` option from the dropdown menu.
5. In the Content pane of the dialog, enter the script or info that you want to have processed by a processing application.
6. Optionally, in the Source field, you can enter a source URI where further information can be made available to the processing application.

Copy XPath of Instance Node

The context menu command, **Copy XPath of Instance Node**, is enabled for elements and attributes defined within a global element or complex type. It copies to the clipboard an XPath expression that locates the selected node.

Find All References and Rename with All References

The two context menu commands, **Find All References** and **Rename with All References**, is enabled for global elements. These, respectively, find and rename the selected global component in the active document and, optionally, in all schema files related to the active document. See [Finding and Renaming Globals](#) for details.

Sorting declarations and references

The **Sort** command is enabled when a node (element or attribute) has multiple declarations or references. The sorting works in the same way as the sorting described in the section, [Schema Overview](#).

Using keyboard shortcuts in Content Model View

You can copy and paste elements in Content Model View using the shortcuts **Ctrl-c** and **Ctrl-v**.

To copy and paste elements:

1. Select the elements you want to copy. To select one element, click on it. To select more than one element, click on the first element and use **SHIFT** and the down arrow key to select further elements.
2. Press **Ctrl-c**. The elements are copied to the clipboard.
3. Select the compositor you want to copy the elements to.
4. Press **Ctrl-v**. The elements are pasted as child elements of the compositor.

To copy and paste elements in reverse order:

1. Click on the lowermost element you want to copy and use **SHIFT** and the up arrow key to select further elements.
2. Press **Ctrl-c**. The elements are copied to the clipboard.
3. Select the compositor you want to copy the elements to.
4. Press **Ctrl-v**. The elements are pasted such that the element that was the uppermost element is now the lowermost element.

About XML Schema annotations

XML Schema annotations are held in the `annotation` element. There are two types of annotation, both of which are elements of the `annotation` element:

- compositor or component documentation, which contains information that could be useful for editors of the schema and is contained in the `documentation` child element of `annotation`.
- application information, which allows you to insert a script or information that a processing application may use; this information is contained in the `appinfo` child element of `annotation`.

Given below is the text of an annotation element. It is based on the example in the description of creating documentation and application information given above.

```
<xs:element name="session_date" type="xs:dateTime" nillable="true">
  <xs:annotation>
    <xs:documentation>Date and time when interview was
held</xs:documentation>
```

```

    <xs:appinfo
source="http://www.altova.com/datehandlers/interviews">separator =
: </xs:appinfo>
    </xs:annotation>
</xs:element>

```

Comments and processing instructions

In XML Schema documents, comments and processing instructions within simple types and complex types are collected and moved to the end of the enclosing object. In such cases, it is therefore recommended that you use annotations instead of comments.

Configuring the content model view

You can configure the content model view for the entire schema in the Schema display configuration dialog (**Schema design | Configure view**). For details about configuration options, see the [Configure view](#) section later in the User Reference. Note that the settings you define here apply to the whole schema, and to the schema documentation output as well the printer output.

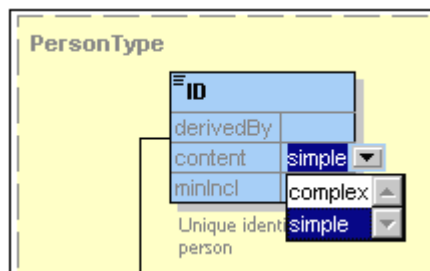
Changing component properties directly in the content model

If the Content Model View is configured so that components are displayed with property descriptor lines (additional information about components) in the component box, then you can edit this information and so change the properties of components. The property descriptor lines

you have defined can be turned on and off by clicking the Add Predefined Details  toolbar icon. You can toggle between a view containing the defined properties and a view not containing them.

To edit component properties:

1. Double-click the (component's) information field that you want to edit, and start entering or editing data. If a predefined option is available, then a drop-down list can be opened and the appropriate entry selected. Otherwise simply enter the required value.



2. Press **Enter** to confirm. The Details entry helpers will be updated to reflect your changes.

Alternatively, you can edit a component's properties in the Details entry helper, and changes will be reflected in the placeholder fields—if these are configured to be displayed.

Documenting the content model

You can generate [detailed documentation](#) about your schema in HTML and MS Word formats. Detailed documentation is generated for each global component, and the list of global components is displayed in a table-of-contents page that allows you to link to the content models of individual components. Additionally, related elements (such as child elements or

complex types) are referenced by hyperlinks, thus enabling you to navigate from element to element. To generate schema documentation, select the menu command **Schema design | Generate documentation**.

Attributes and Identity Constraints

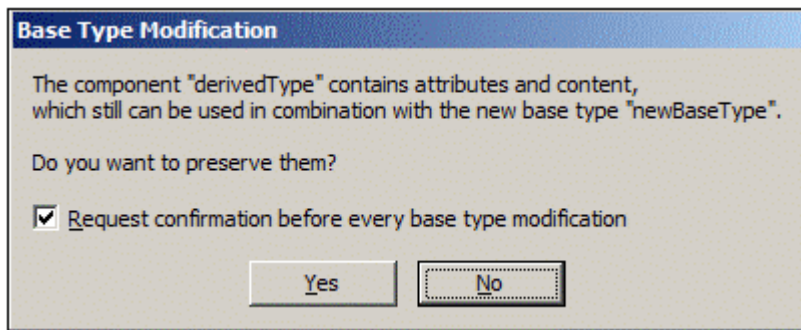
Attributes and Identity constraints can appear in a pane below the Content Model View or as boxes in the Content Model View itself. You can toggle between these views by clicking the  icon. For a description of how to insert and edit attributes and identity constraints, see [Defining attributes for components](#) and [Defining identity constraints for components](#), respectively.

4.3.3 Base Type Modification

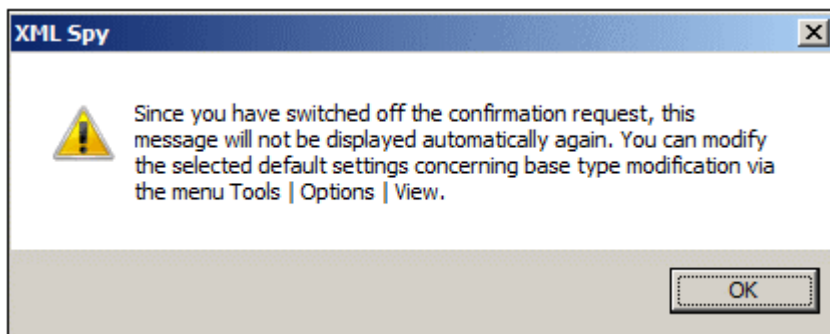
If the base type of a derived type is changed in Schema View, content, attributes, facets and sample values defined within the derived type can be handled in one of two ways:

- They can be preserved if they are still applicable in combination with the new base type.
- They can be removed automatically whether or not they are still applicable in combination with the new base type.

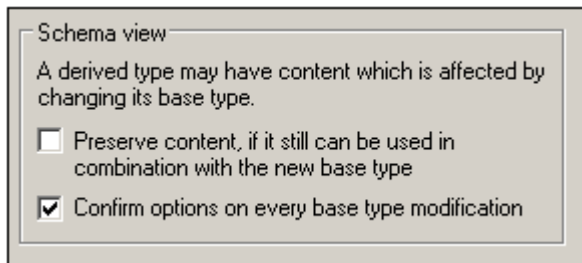
When changing the base type of a derived type which contains content, attributes, facets or sample values the Base Type Modification dialog (*screenshot below*) is displayed.



If the Request Confirmation check box is de-selected a pop-up (*screenshot below*) indicates that the confirmation can be turned on again in the View tab of the Options dialog ([Tools | Options | View](#)).



In the Schema View pane (*screenshot below*) of the View tab of the Options dialog ([Tools | Options | View](#)), you can specify whether content should be preserved and whether user confirmation is required for every base type modification.



Check the respective check boxes to preserve content and require confirmation if you wish these to be the default options.

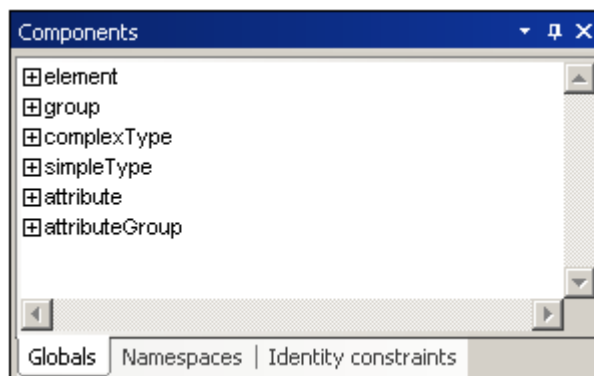
4.3.4 Entry Helpers in Schema View

There are three Entry Helpers in Schema View: Component Navigator, Details Entry Helper, and Facets Entry Helper. The entry helpers are the same in both Schema Overview and Content Model View. They are described below.

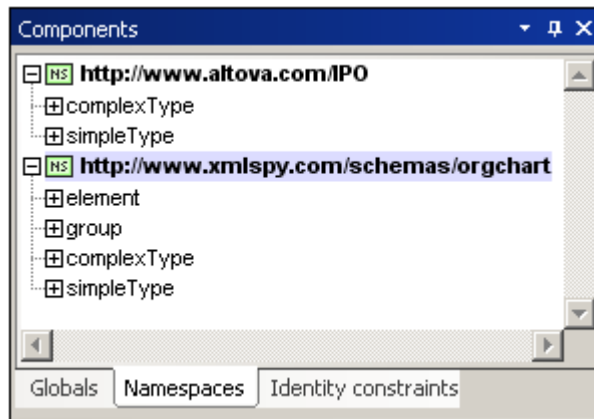
Component Navigator

The Component Navigator is an Entry Helper in **Schema View**. It serves three purposes:

- To organize global components in a tree view by component type and namespace (see *screenshots below*). This provides organized overviews of all global components.
- To enable you to navigate to and display the Content Model View of a global component—if the component has a content model. If a component does not have a content model, the component is highlighted in the Schema Overview. Global components that are included or imported from other schemas are also displayed in the Component Navigator.
- To provide an overview of the identity constraints defined in the schema document. For a description of the Identity Constraints tab, see [Identity Constraints Overview](#).



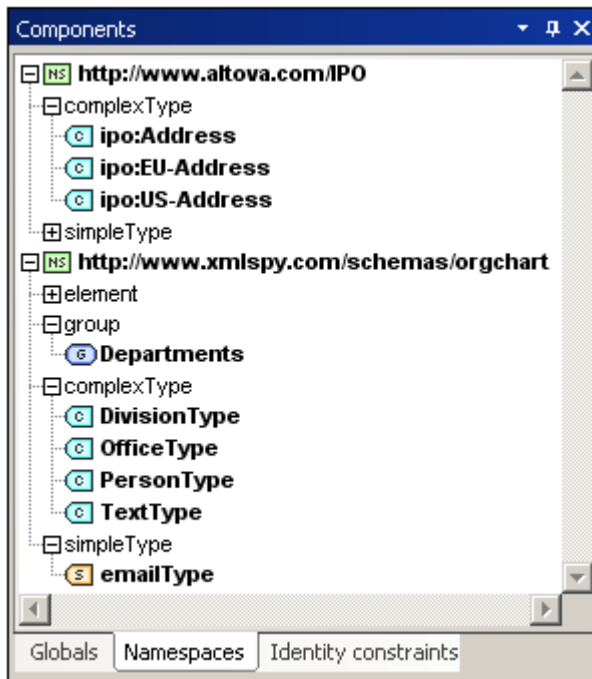
In the Globals tab (*above*) global components are grouped in a tree according to their component type. In the Namespace tab (*below*), components are organized first according to namespace and then according to component type. Note that a component type is listed in a tree only if at least one component of that type exists in the schema.



In the tree display, global components are organized into the following six groups:

- Element Declarations (Elements)
- Model Groups (Groups)
- Complex Types
- Simple Types
- Attribute Declarations (Attributes)
- Attribute Groups

Expanding a component-type group in the tree displays all the components in that group (see *screenshot*). This enables you to easily navigate to a required component.

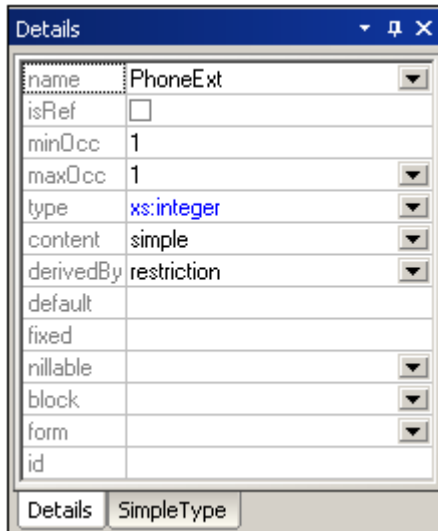


If a component has a content model (i.e., if it is an Element, Group, or Complex Type), double-clicking it will cause the content model of that component to be displayed in Content Model View (in the Main Window). If the component does not have a content model (i.e. if it is a Simple Type, Attribute, or Attribute Group), then the component is highlighted in the Schema Overview (in the Main Window).

Please note: If the component is in an included or imported schema, then the included/imported schema is opened (if it is not already open), and either the component's content model is displayed in Content Model View or the component is highlighted in Schema Overview.

Details Entry Helper

The Details Entry Helper is available in **Schema View**. It displays editable information about the compositor or component currently selected in the Main Window. If you are editing a schema file which contains database extensions, an additional tab with information about the DB extensions may be visible.



The screenshot shows the 'Details' dialog box with a list of properties for a component named 'PhoneExt'. The properties and their values are:

Property	Value
name	PhoneExt
isRef	☐
minOcc	1
maxOcc	1
type	xs:integer
content	simple
derivedBy	restriction
default	
fixed	
nilable	
block	
form	
id	

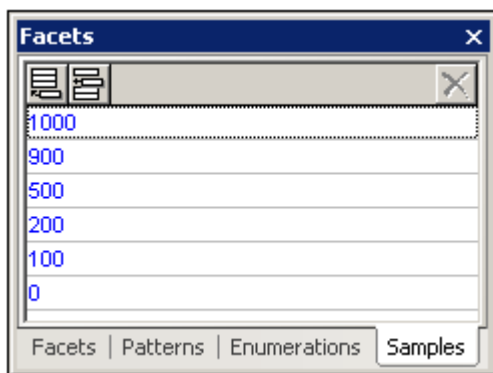
At the bottom, there are two tabs: 'Details' (selected) and 'SimpleType'.

To change the properties of the currently selected compositor or component, double-click the field to be edited and edit or enter text directly. If a combo box is available in the field to be edited, select the desired value; this value is entered in the field.

Changes you make via the Details Entry Helper are immediately reflected in the content model diagram.

Facets Entry Helper

The Facets Entry Helper is available in the **Schema View**, and enables you to enter the values of facets, patterns, sample values, and enumerations.



The screenshot shows the 'Facets' dialog box with a list of values for a facet. The values are:

Value
1000
900
500
200
100
0

At the bottom, there are four tabs: 'Facets' (selected), 'Patterns', 'Enumerations', and 'Samples'.

To change facets, patterns, Sample Values, or enumerations in the Facets Entry Helper:

1. Select the required tab (Facets, Patterns, Sample Values, or Enumerations)
2. If a combo box is present, select a value from the drop-down menu. Alternatively, double-click a row, and edit or enter text directly.

Note:

- You can use the cut, copy and paste shortcuts (CTRL+X, CTRL+C, CTRL+V, respectively) to copy the patterns and enumerations of one component to another component. In the Facets Entry Helper, select the pattern/s or enumeration/s to copy, cut or copy the selection, then click in the Facets Entry Helper window of the target component, and paste.
- Sample values can be used when generating an XML file from the schema (with the menu command **DTD/Schema | Generate Sample XML File**). If a sample value is invalid, this is flagged as a warning and the sample value is displayed in orange in the Samples tab. You can switch off the validation warnings icon  in the toolbar. Note also that, in the case of invalid samples, the XSD file itself is not affected; it will still be valid if the file is valid in other respects.

4.3.5 Identity Constraints

Identity constraints can be defined for a global component via two entry points:

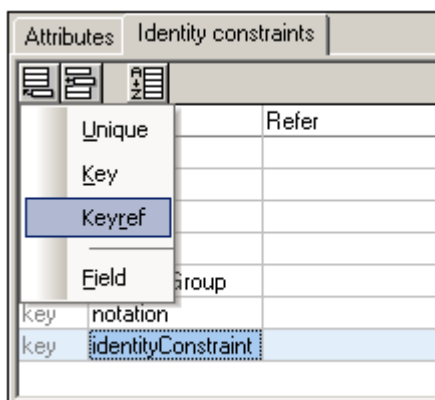
- In [Schema Overview](#), select a global component and define identity constraints in the Identity Constraints tab (at the bottom of the view and below the main pane).
- In the [Content Model View](#) of a global component. Content Model View provides a graphical representation of the identity constraints and drag-and-drop editing functionality, neither of which is available when editing identity constraints in Schema Overview.

In this section, we describe these two mechanisms for creating and editing identity constraints. An [overview of all identity constraints](#) in the schema is available in the Identity Constraints tab of the Components entry helper; this is described [further below](#).

Identity constraints in Schema Overview

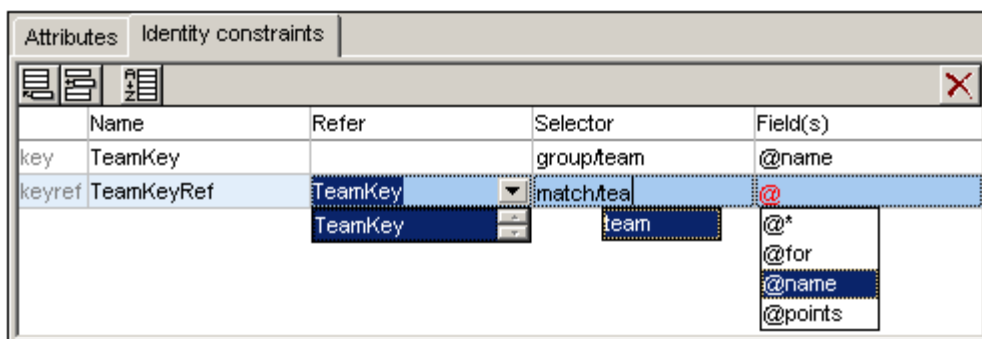
To define identity constraints for a global component in Schema Overview, do the following:

1. Select the global component in the global components list of [Schema Overview](#).
2. In the Attributes/Identity Constraints pane, select the Identity Constraints tab.
3. Click the **Append**  or **Insert**  icon at the top left of the Identity Constraints tab.
4. From the popup that appears (*screenshot below*), select the type of identity constraint you want to append or insert: *Unique*, *Key*, or *Keyref*.



An entry is created in the Identity Constraints list.

5. In the newly created entry, enter the Identity Constraint's properties. Give the identity constraint a name (see screenshot below) and then define XPath expressions for the selector and field elements. For each location step in these XPath expression, a pop-up containing the available nodes appears (see screenshot below). For the refer element of the keyref kind of identity constraint, a dropdown list shows the available key elements (see screenshot below).



If you wish to define multiple field elements on an identity constraint, select the identity constraint, click the **Append** or **Insert** icon (see Step 3 above), and select **Field**. An empty Field entry box is added to the selected identity constraint. Enter the required XPath expression in the Field entry box.

Note: An ID can be assigned to any of these elements (that is, to the identity constraint, its selector, and/or field/s). To assign an ID, select the element and, in the Details entry helper, enter the ID in the `id` row.

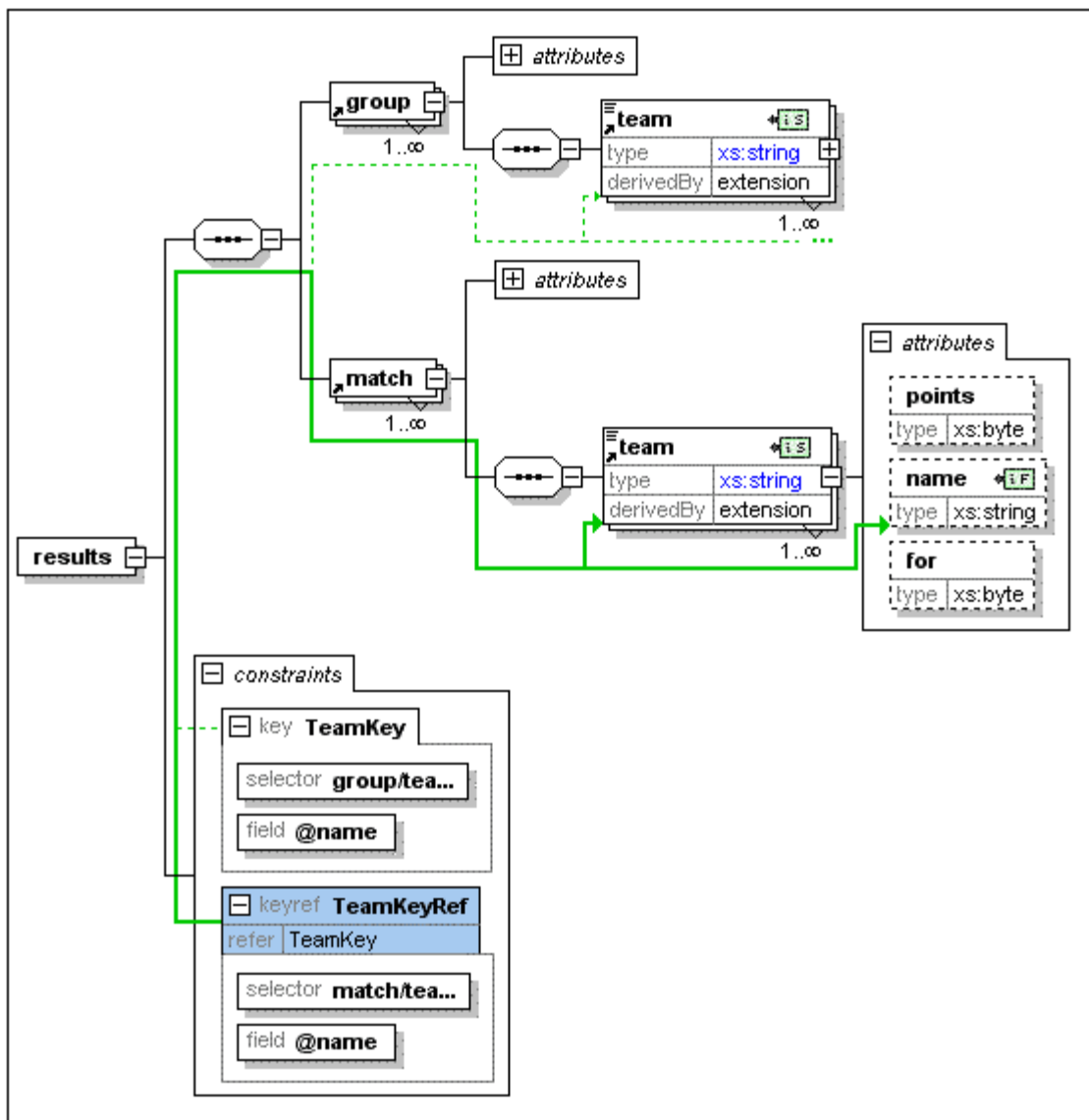
Identity constraints in Content Model View

To view identity constraints in [Content Model View](#), do the following:

1. In Schema Overview, select the global component for which you wish to create the identity constraint and switch to the [Content Model View](#) of that component.
2. Ensure that the display of identity constraints is toggled on in the Schema Display Configuration dialog (**Schema Design | Configure View**). This toggle switches on and off the display of the identity constraints box in the design (see screenshot below). You can also toggle on and off the display of the Constraints box with the Display Constraints  icon in the Schema Design toolbar.
3. Ensure that the Visualize ID Constraints icon  in the Schema Design toolbar is toggled on. This ensures that relationship lines and ID constraint information are displayed in Content Model View (see screenshot below).

Note: The Visualize ID Constraints icon  switches on ID Constraint editing and validating functionality in Schema View. If an XPath expression in an ID Constraint does not locate a node, an error or warning might be displayed. This error or warning is to alert you to the incorrect XPath expression; the XML Schema document itself is not invalid because of the incorrect XPath expression. To re-check the validity of the document without the XPath expression check, switch to Text View or Grid View and validate. You can also disable validation in Schema View by toggling the Visualize ID Constraints icon  off.

The screenshot below shows the graphical display of identity constraints. There are two identity constraints in the identity constraints box: `TeamKey` and `TeamKeyRef`. The properties of `TeamKeyRef` are highlighted with a solid green line (since `TeamKeyRef` is selected). The properties of `TeamKey` (unselected) are shown with a dotted green line. For each identity constraint the node selected by the XPath expression for `selector` and `field` are shown with the icons  and  respectively. If a node is collapsed, as is the case with the `field` element of the `TeamKey` constraint, the relationship line ends with an ellipsis (see screenshot below).



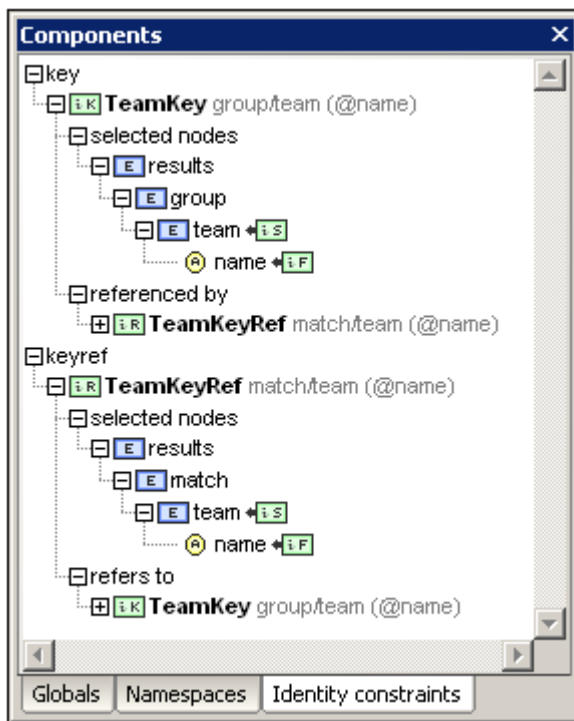
To create an identity constraint in [Content Model View](#), do the following:

1. In the Constraints box, right-click the Constraints entry or the name of an existing identity constraint.
2. Select **Insert** or **Append**, and then select the kind of identity constraint to insert (at the top of the list of existing constraints) or append (at the bottom of the list of existing constraints). The identity constraint will be created with an empty `selector` element and an empty `field` element.
3. The XPath expression can be entered in the `selector` and `field` element boxes in one of two ways: (i) By typing it in, or (ii) By dragging the target node into the `selector` or `field` element box and dropping it when the box becomes highlighted; the XPath expression will be created automatically.
4. To add an additional `field` element to an identity constraint, either right-click the constraint name and select **Add child | Field** from the context menu, or right-click the `selector` or `field` element and select **Insert | Field** or **Append | Field** from the context menu.
5. To enter an ID attribute on any element in a constraint, select that element and enter a value in the `id` row of the Details entry helper.

Note: Identity constraint information can be toggled on and off with the Enable ID Constraints Information icon  in the Schema Design toolbar. The display of the entire Constraint box can be toggled on and off in the Schema Display Configuration dialog (**Schema Design | Configure View**) or with the Display Constraints  icon in the Schema Design toolbar.

Identity constraints overview

The Identity Constraints tab of the Components entry helper (*screenshot below*) provides an overview of a document's identity constraints. In this tab, identity constraints are listed by the kind of identity constraint (`unique`, `key`, `keyref`) and displayed as an expandable/collapsible tree.



Entries in bold are present in the current schema, while those in normal face are present in sub-schemas. Double-clicking an entry in the Identity Constraints tab selects that schema component in [Content Model View](#).

The following context menu commands are available when an item in the Identity Constraints tab is selected:

- *Show in Diagram*: selects the schema component in [Content Model View](#).
- *Show Selector/Field Target in Diagram*: selects, in [Content Model View](#), the schema component targeted by the selector or field of the identity constraint. In the case of multiple fields, a dialog prompts the user for the required field.
- *Go to Identity Constraint*: selects the identity constraint in [Schema Overview](#).
- *Expand/Collapse All*: expands or collapses the tree, respectively.

4.3.6 Smart Restrictions

When restricting a complex type, parts of the content model of the base type are rewritten in the derived type. This can be confusing if the content model is complex because while editing the derived type it might be hard to correctly remember exactly what the content model of the base type looks like.

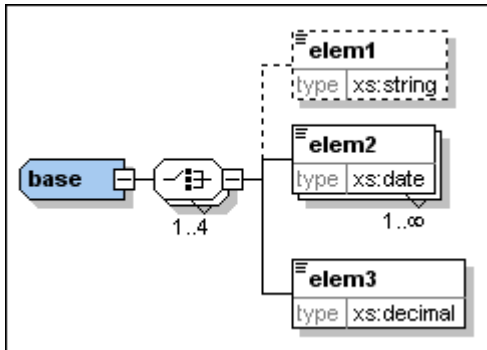
Smart Restrictions combine and correlate the two content models in the graphical view of the derived content model. In the derived complex type, all particles of the base complex type, and how they relate to the derived type, can be seen. Additionally, Smart Restrictions provide visual hints to show you all possible ways to restrict the base type. This makes it easy to correctly restrict the derived type.

To switch on Smart Restrictions:

- Click the Smart Restrictions icon  in the Schema Design toolbar.

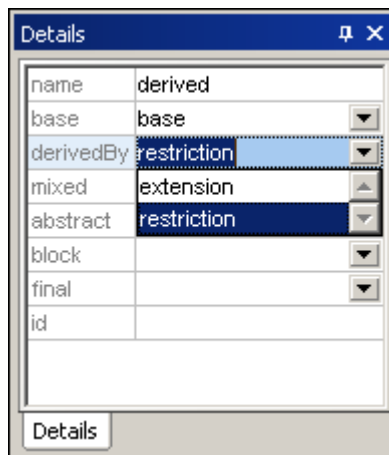
The example that follows illustrates the features of Smart Restrictions.

The following complex type is the base type used in this example:

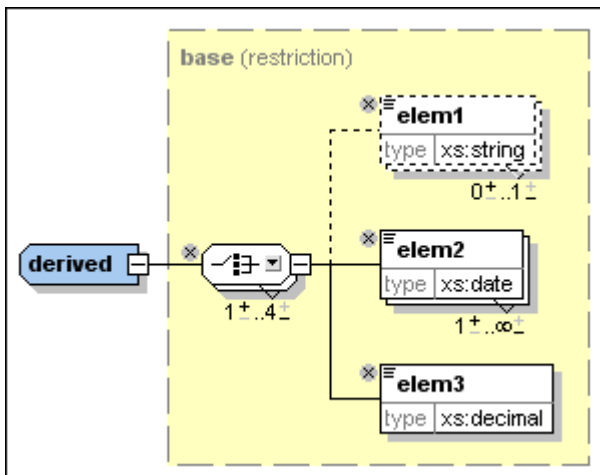


The complex type "derived" is derived from the "base" type as follows:

1. Create a new complex type in the schema and call it "derived".
2. In the Details Entry Helper select "base" from the **base** drop-down list and "restriction" from the **derivedBy** drop-down list.

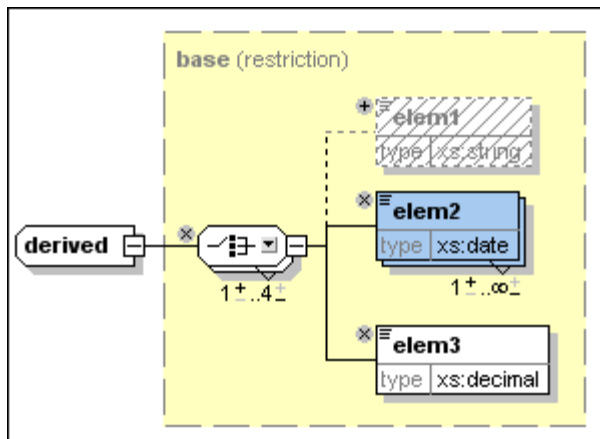


With Smart Restrictions switched on, the new derived type looks like this:

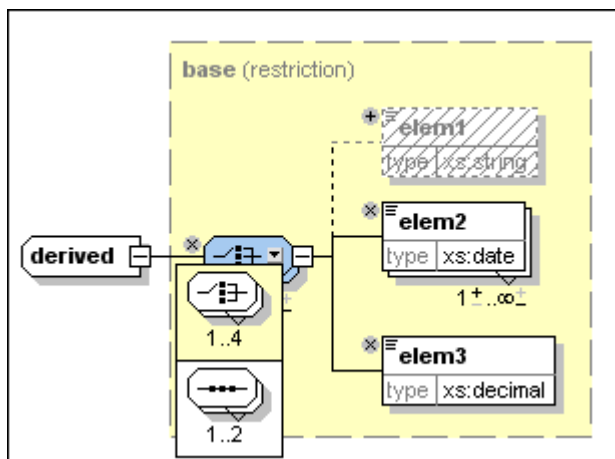


Notice the following controls that can be used to restrict the derived type in this example:

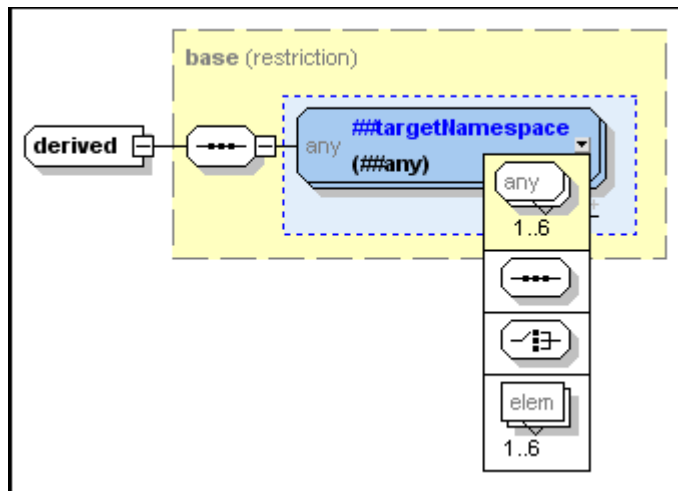
- Use this icon  to remove elements that are in the base type from the derived type. Here, elem1 has been deleted. To add it again, click this icon .



- Click the down arrow on the Choice compositor  to get the following list, which allows you to change the Choice model group to a Sequence model group:

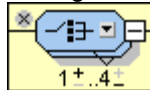


It is also possible to change wildcards in the same way, as seen in this example:

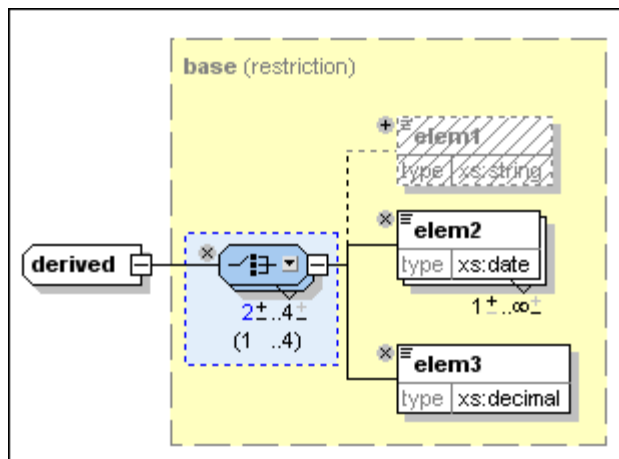


For a complete list of which particles can be replaced by which other particles, see the [XML schema specification](#).

- Change the number of occurrences of the model group using the following control

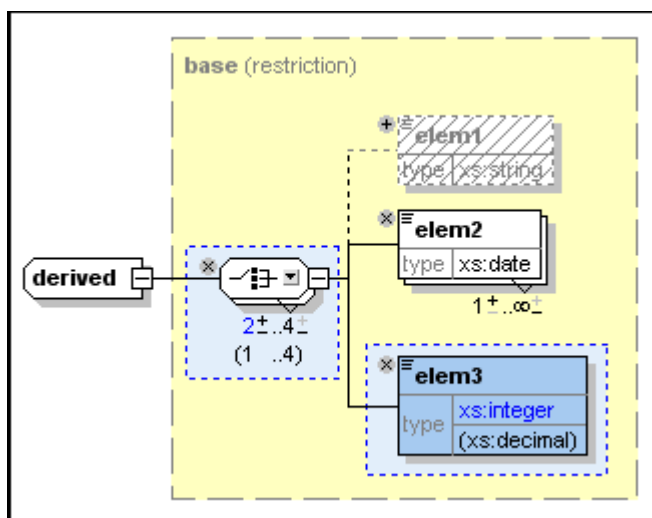


to increase the minimum number of occurrences by clicking the plus sign over the "1", or to decrease the maximum number of occurrences by clicking the minus sign under "4". These controls are shown if the occurrence range in the base describes a real range (e.g., 2-5) and not a certain amount (e.g. 4-4). They are also displayed if the occurrence range is wrong.

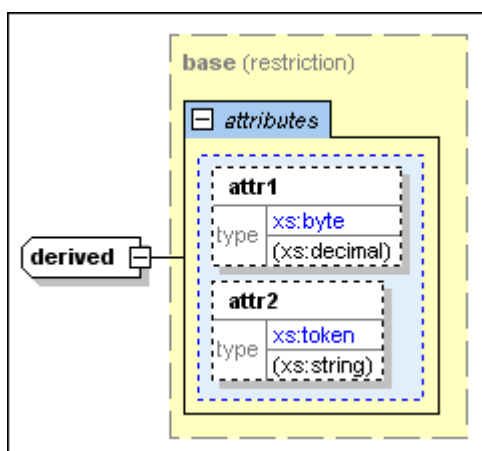


Here you can see that the minimum occurrence for this element has been changed to 2. Notice that the model group now has a blue background, which means that it is no longer the same as the model group in the base complex type. Also, the permitted occurrence range of the model group in the base particle is now displayed in parentheses.

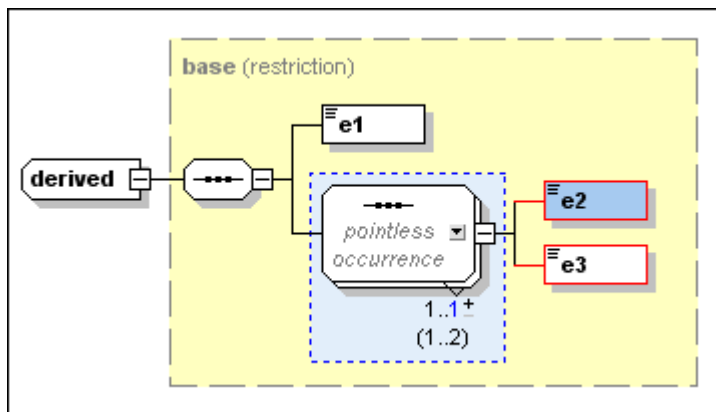
- It is possible to change the data types of attributes or elements if the new data type is a valid restriction of the base data type as defined in the [XML schema specification](#). For example, you can change the data type of elem3 in the "derived" data type from decimal to integer. After you do this, the element has a blue background to show that it is different from the element in the base type, and the type that the element has in the base type is displayed in parentheses:



This example shows attributes whose data types have been restricted in the derived complex type:



- Smart Restrictions alert you to *pointless occurrences* in the content model. A pointless occurrence happens, for example, when a sequence that is present in the content model is unnecessary. This example shows a pointless occurrence:



Please note: Pointless occurrences are only shown if the content model contains an error. It is possible for a content model to contain a pointless occurrence and be valid,

in which case the pointless occurrence is not explicitly shown in order to avoid confusion.

See the [XML schema specification](#) for more information about pointless occurrences.

4.3.7 xml:base, xml:id, xml:lang, xml:space

The namespace `http://www.w3.org/XML/1998/namespace` is, [according to the XML Namespaces specification](#), bound by definition to the `xml:` prefix. What this means is that this is the namespace that must be used with the `xml:` prefix and that is reserved for it. There are four attributes in this namespace that can be children of any XML element in any XML document (schema or instance):

- `xml:base` (for setting the base URI of an element)
- `xml:id` (for specifying the unique ID of an element)
- `xml:lang` (for identifying the language used within that element)
- `xml:space` (for specifying how whitespace in the element should be handled)

In Schema View, once the XML Namespaces namespace has been imported into the XML Schema document, these four `xml:` attributes can be referenced for use on any element in the schema.

In order to declare one of these attributes on an element, do the following:

1. Declare the XML Namespaces namespace for that schema document and bind the namespace to the `xml:` prefix. When any of the four `xml:` attributes is used in the document, its name would then be expanded to include the correct namespace part.
2. Import the XML Namespaces namespace. XMLSpy's validator will recognize the namespace and make the four `xml:` attributes available as global attributes, which can be referenced within that schema.
3. Insert the required `xml:` attribute as the child of an element. The attribute is declared as a reference to the "imported" global attribute.

Declare the XML Namespaces namespace

You can declare the XML Namespaces namespace (`http://www.w3.org/XML/1998/namespace`) by entering it via the Schema Settings dialog, where all namespaces declared for that schema are stored and can be edited. The namespace must be bound to the `xml:` prefix. (Alternatively, you could declare the namespace (with the `xml:` prefix) on the `xs:schema` element in Text View.)

Import the XML Namespaces namespace

In Schema Overview, create a global import declaration for the XML Namespaces namespace.

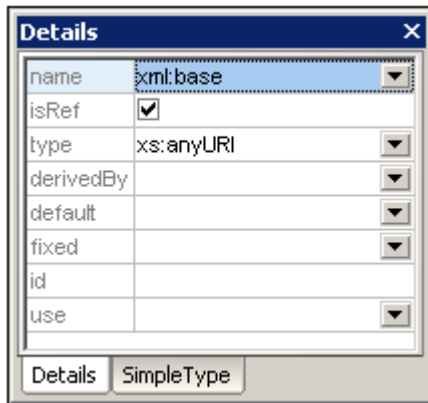
Do this by clicking the Insert  or Append  icon at the top of the Schema Overview window and selecting **Import** from the menu that pops up. Enter the XML Namespaces namespace as the namespace to be imported. In Text View, the import declaration should look like this:

```
<xs:import namespace="http://www.w3.org/XML/1998/namespace"
schemaLocation="http://www.w3.org/XML/1998/namespace"/>.
```

Adding the `xml:` attribute

In Schema Overview, select the element for which the `xml:` attribute is to be added, and add an attribute for it. In the Details entry helper (*screenshot below*), click the down arrow of the name

combo box and select the required `xml:` attribute, for example `xml:base`. When you are prompted whether you wish to reference the global attribute, click **Yes**. The attribute is added as a reference.



XInclude and `xml:base`

When XInclude's `include` element is replaced by the XML file specified in the `href` attribute of the include element, the top-level element of the parsed XML document is included with an `xml:base` attribute. If this XML document is going to be validated, then the schema must define an `xml:base` attribute on the relevant element/s.

4.3.8 Back and Forward: Moving through Positions

The **Back** and **Forward** commands in Schema View enable you to move through previously viewed positions in Schema View. This is useful because, while clicking through schema components in Schema View, you might wish to view a previously viewed component. Clicking the **Back** button once in the toolbar takes you to the previously viewed position. By repeatedly clicking the **Back** button, you can view up to 500 of the last visited positions. After moving back through previous positions, you can move forward through these positions by using the **Forward** button in the toolbar.

The shortcut keys for the two commands are:

-  **Back: Alt + Left Arrow**
-  **Forward: Alt + Right Arrow**

Back/Forward versus Undo/Redo

Note that the **Back** and **Forward** commands are not the same as the **Undo (Ctrl+Z)** and **Redo (Ctrl+Y)** commands. These two sets of commands make up two different series of steps. Clicking the **Back** command once takes you to the previously viewed component as previously displayed. Clicking the **Undo** command once undoes the last editing change regardless of when that editing change was made.

Additional notes

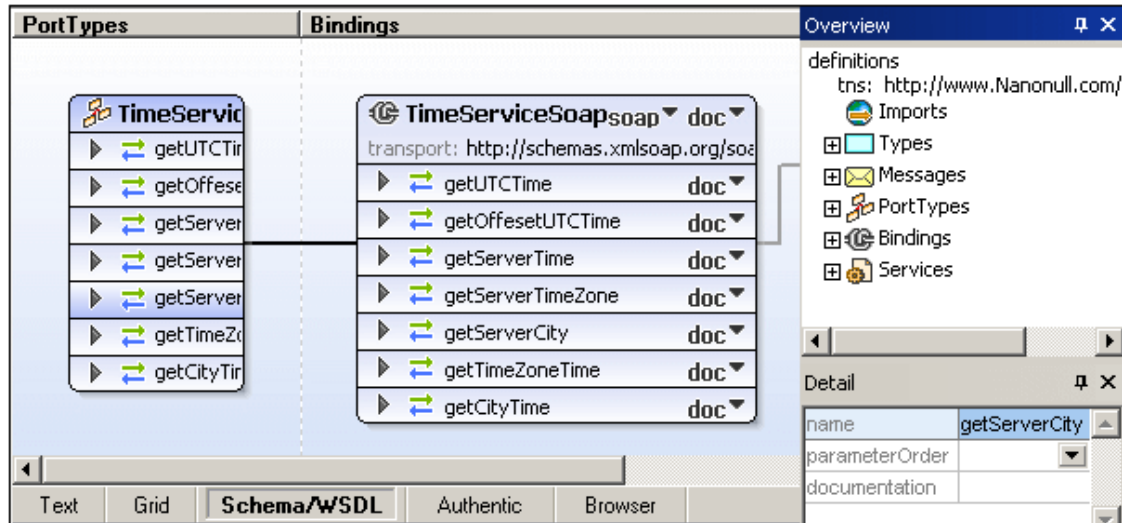
Note the following points:

- The **Back** button enables you to re-view the previous 500 positions.

- The Back/Forward feature is enabled across schemas. If a schema has since been closed or is currently open in another view, it will be opened in Schema View or switched to Schema View, respectively.
- If a component that was viewed in a previous position is deleted, then that component will not be able to be viewed. If such a component was part of a previous position, this position will be displayed without the deleted component. If the component comprised the entire position, the entire position will be unavailable, and clicking the **Back** button at this point in the Back series will take you to the position previous to the unavailable position.

4.4 WSDL View

WSDL View (*screenshot below*) provides an interface for graphically editing WSDL 1.1 and WSDL 2.0 documents. WSDL View is available when a WSDL document is active and the WSDL View tab is clicked. The structure and components of a WSDL document are created in the [Main Window](#) using graphical design mechanisms, and additional editing is enabled from the [Entry Helpers](#).



The [Main Window](#) (consisting of the PortTypes (WSDL 1.1) or Interfaces (WSDL 2.0), Bindings, and Services sections) and the Entry Helpers ([Overview](#) and [Details](#)) are described in the sub-sections of this section. (For a description of how to work with projects, see [Project Menu](#) in the User Reference section.)

Functionality available in WSDL View

The following functionality is available in WSDL View:

- A graphical display in the Main Window of all WSDL elements, grouped by PortTypes (WSDL 1.1) or Interfaces (WSDL 2.0), Bindings, and Services.
- Direct manipulation of WSDL elements using drag and drop.
- Ability to add, append, and delete any WSDL element visible in the graphical view (context sensitive menu).
- Ability to enter and edit values in the Details Entry Helper.
- WSDL validation against W3C Working Draft.
- Automatic switching to Schema View if a referenced or imported schema is found to be invalid.
- Editing of schema types from within WSDL View.
- Generation of WSDL documentation in MS Word or HTML.
- Generation of a diagram (PNG image) of the WSDL document in the Main Window.
- Printing of the view in the WSDL window.

File viewing

Note the following points concerning file viewing:

- When you open a WSDL file, the file opens automatically in WSDL View.
- You can also view a WSDL document in the Text and Enhanced Grid Views. To do this, click on the appropriate tab.

- If the WSDL file contains a reference to an XML Schema, then the schema can be viewed and edited by selecting the menu command **WSDL | Types | Edit Schema in Schema View**. This opens the schema file in the Schema View.
- If an associated schema file is open, then you are not allowed to change the view of the WSDL file (for example, from WSDL View to Text View). Before trying to change views of the WSDL file, make sure that you have saved changes to the schema file and closed the file.

There are two entry helpers to help you edit WSDL documents: [Overview](#) and [Details](#). Both entry helpers can be docked/undocked by double-clicking the title bar. When docked, the auto-hide feature can be activated by clicking the drawing-pin icon in the title bar. When auto-hidden, the entry helper is minimized as a tab at an edge of the application window. An auto-hidden entry helper can be re-docked by rolling it out from the edge (by mousing over its tab) and clicking the drawing-pin icon in the title bar.

4.4.1 Main Window

The Main Window is where you edit your WSDL document. It consists of three vertical sections: (i) [Port Types \(WDL 1.1\) or Interfaces \(WSDL 2.0\)](#); (ii) [Bindings](#), and (iii) [Services](#). The relationship between a port type and a binding and between a binding and a service is each indicated with a connector line. Each of these three sections is described in detail below.

Symbols in the Main Window

The following symbols are used in the Main Window:



Port type in WSDL 1.1, Interface in WSDL 2.0



Binding



Service



Fault



Operation. The green arrow represents inputs and the blue arrow represents outputs. Depending on the type of operation, an appropriate symbol is used.



Message



Message part (parameter)



XSD element



XSD simpleType or complexType



Port

Adding new port types, interfaces, bindings, and services

To add a new port type (in WSDL 1.1 documents), interface (in WSDL 2.0 documents), binding, or service, right-click anywhere in the Main Window but outside a component box, and select the relevant command from the context menu that appears.

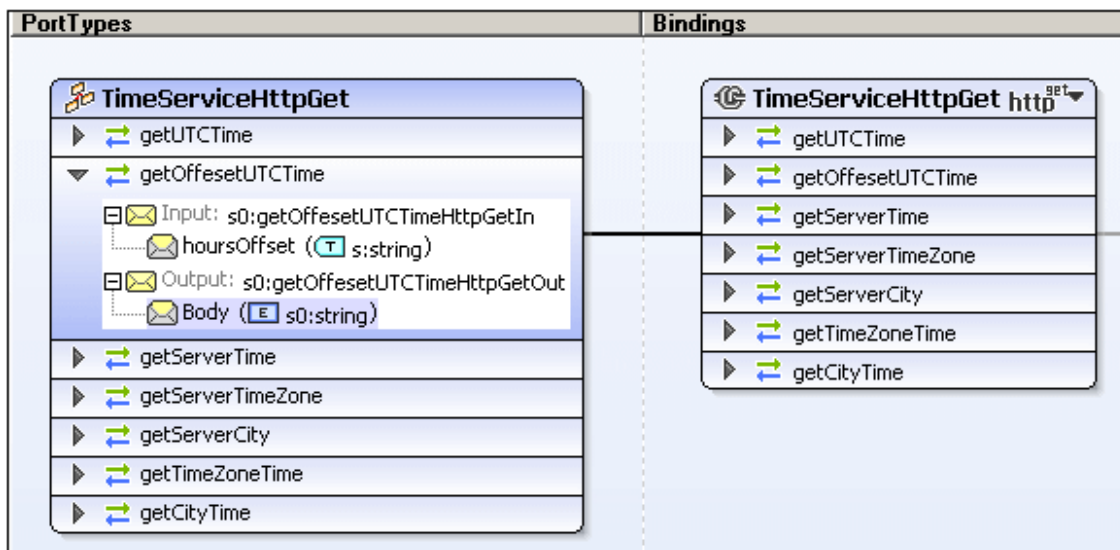
Drag and drop functionality

The following drag-and-drop functionality is available:

- In the Main Window, associations between PortTypes (WSDL 1.1) or Interfaces (WSDL 2.0) and Bindings and between Bindings and Services can be established with drag-and-drop.
- In WSDL 2.0 documents, elements in the Overview entry helper can be dragged to interface faults in both the Main Window and the Overview entry helper.

PortTypes (WSDL 1.1), Interfaces (WSDL 2.0)

The PortTypes section (WSDL 1.1 documents) contains all the portTypes defined in the WSDL document (*the screenshot below shows only one portType in the PortTypes section*). The Interfaces section (in WSDL 2.0 documents) contains all the interfaces defined in the WSDL document



Each portType or interface  is represented as a box containing the operations  defined for that portType or interface. Components can be edited directly in the box. The main features of port type and interface boxes are listed below:

- Operations can be expanded to display their messages  by clicking the  icon at the left of an operation name.
- In WSDL 1.1 a message can contain a message part . Such messages can be expanded to show the message part.
- Right-clicking a component of a portType box (either portType, operation, message, or message part), pops up a context menu from which relevant actions can be selected. For example, right-clicking a portType name allows you, among other actions, to append a new portType, append an operation to the selected portType, or create a binding for the selected portType.
- The optional WSDL 2.0 interface properties, *extends*, *styleDefault*, and *documentation* are hidden if empty. They can be edited via the **Edit** command in the context menu of the interface.
- In WSDL 2.0 documents, properties of operations can be edited via the **Edit** command in the context menu of the operation. The value of the style property is selected via a

- combo box listing the options.
- Note that when a component is selected, its details can be edited in the Detail entry helper.
- Documentation for port types and interfaces appears at the bottom of individual boxes.

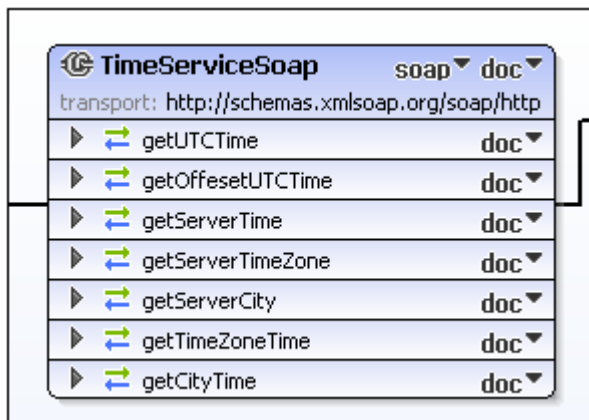
The association of a portType or interface with a binding is indicated in the Main Window by a black connector line linking the portType box or interface box to the binding box; the binding box will be in the Bindings section of the Main Window.

Bindings

A binding defines message formats and protocol details for:

- Operations defined by a particular portType (WSDL 1.1), or
- Operations and faults defined by a particular interface (WSDL 2.0).

In WSDL 1.1, bindings can be created for SOAP 1.1 or SOAP 1.2 endpoints, or for HTTP 1.1's GET and POST verbs. In WSDL 2.0, bindings can be created for SOAP 1.1 or SOAP 1.2 endpoints, or for HTTP. Each binding is represented by a binding box (*screenshot below*) in the Bindings section of the Main Window. The binding box contains all the operations and/or faults of the associated portType or interface (*see screenshot below*).



A binding can be associated with a port type or an interface in any of the following ways:

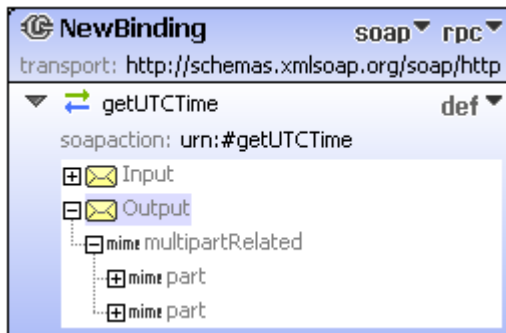
- Right-click a port type or an interface and select the command **Create binding for portType** or **Create binding for interface**, respectively.
- Right-click a WSDL 1.1 binding and edit the *PortType* property.
- Right-click a WSDL 2.0 binding and select the command **Edit | Interface**.

To define the binding, in the first combo box to the right of the binding name (*screenshot below*), select the required protocol. In WSDL 1.1, this is either `soap 1.1`, `soap 1.2`, `http-get`, or `http-post` to define the kind of binding. If you select a SOAP protocol, you can additionally define (using the second combo box) whether the style should be `doc` or `rpc`. In WSDL 2.0 documents, the `wsa: protocol` property can be added or edited via the **Edit** command of the context menu of the binding.



In WSDL 1.1, MIME encodings (also referred to as MIME bindings) are defined at the message

level. To define a MIME encoding, right-click on the message (*screenshot below*) and append the appropriate MIME definition. In the screenshot below, MIME definitions have been created for the `Output` message.



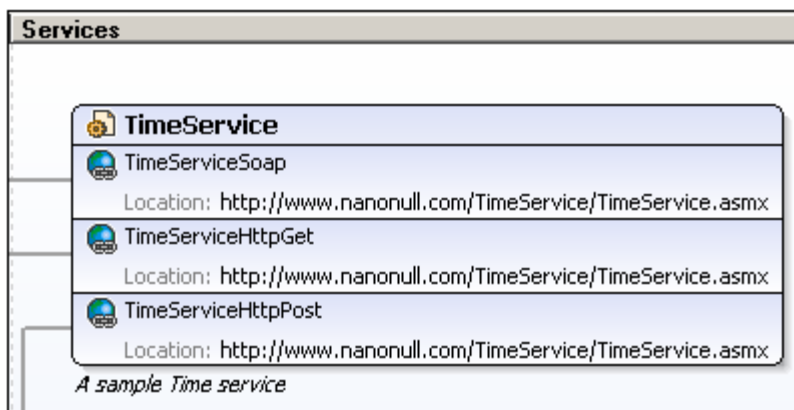
Right-clicking a specific item in a binding box opens a context-sensitive menu. Using the context menus, for example, bindings can be appended or deleted; extensibility items can be edited; and messages defined. Note also that when a binding box or an item in a binding box is selected, the definitions are displayed in the Details entry helper and can be edited there.

A port can be created for a binding by right-clicking the title bar of a binding box and selecting the **Create Port for Binding** command (WSDL 1.1 documents) or **Create Endpoint for Binding** command (WSDL 2.0 documents). The associated port or endpoint is created within a service box (in the Services section of the Main Window). The association between a binding and a port is indicated by a black connector line.

Documentation for bindings appears at the bottom of individual binding boxes.

Services

A service groups together a set of related ports (WSDL 1.1) or endpoints (WSDL 2.0). It is represented by a service box in the Services section of the Main Window (*screenshot below*). Each service box consists of one or more port or endpoint declarations (*see screenshot below*).

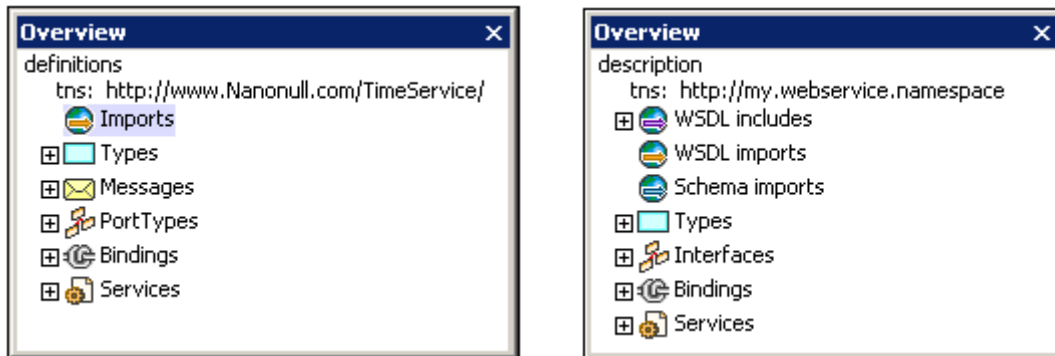


The service name, port or endpoint name, the binding associated with a port or endpoint, and the address information of a port or endpoint can be edited directly in the service box or in the Details entry helper. Right-clicking a service box or a specific item in the service box opens a context menu in which commands relevant to the service or that item are available.

Documentation for services appears at the bottom of individual service boxes.

4.4.2 Overview Entry Helper

The **Overview entry helper** (screenshot below) provides an overview of the WSDL document by grouping the document's various components into structural categories and by listing the target namespace, imported schemas, and included/imported WSDL documents. In addition to port types (or interfaces in WSDL 2.0), messages (WSDL 1.1), bindings, and services, the various types defined in the document are also listed.



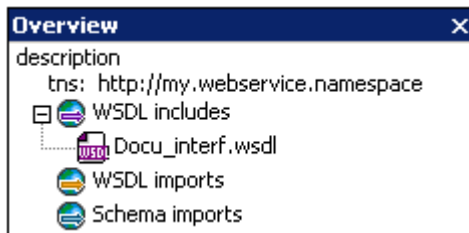
Overview entry helper in WSDL 1.1 (left) and WSDL 2.0 (right).

In each category, components are displayed in a tree view. A tree item can be expanded and collapsed, respectively, to reveal and to hide its contents. Selecting a component in the Overview entry helper displays it and its properties in the [Details entry helper](#), where the properties can be edited. The names of WSDL and schema components that are displayed in the tree can be edited directly in the trees. Externally defined components (those in included or imported WSDL documents or schemas and displayed in gray), however, cannot be edited. The individual categories in the Overview entry helper are explained below.

Target namespace (WSDL 1.1 and 2.0): Indicated in the tree by `tns`. The target namespace can be edited in the Overview entry helper. All other namespaces must be edited in Text View.

Imports (WSDL 1.1): XML Schema (XSD) files and WSDL files can be imported into the active WSDL document. To import a file, right-click the Imports item or an already imported file in the Imports list, and select **Add new import**. Right-clicking an imported file in the Imports list pops up a context menu in which you can choose to add a new import, select another file to replace the selected file as an import (**Edit import**), or delete the imported file. You can also open the file from its location. The file opens in WSDL View (`.wsdl` file) or Schema View (`.xsd` file), and can be edited there.

WSDL includes, WSDL imports, Schema imports (WSDL 2.0): XML Schema (XSD) files can be imported, and WSDL files can be included or imported, into the active WSDL document. To include or import a file, right-click the respective item and add the include or import. The namespace of an imported file is generated automatically from the target namespace of the imported file.



Right-clicking an included or imported file pops up a context menu in which you can choose to delete the file or open it in XMLSpy. The file opens in WSDL View (.wsdl file) or Schema View (.xsd file), and can be edited there.

Types (WSDL 1.1 and 2.0): Lists all types defined in the WSDL document (in black) and in any imported schema or WSDL document (in gray). You can create a new XSD element, simpleType, or complexType by right-clicking any type in the list and selecting the appropriate Add New command. A newly created type is added to the WSDL `types` definitions and is listed under the Types item in the Overview entry helper.

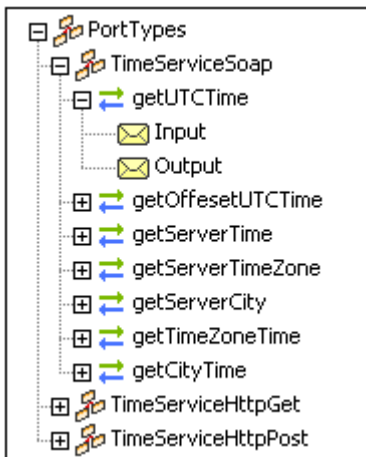


A type defined in the WSDL document can be edited in XMLSpy's Schema View. To do this, right-click the type you wish to edit and select Edit (Schema View) from the context menu that pops up. This causes a temporary XSD file to be generated on the fly from the `types` definitions in the WSDL document. This XSD document is displayed in Schema View and can be edited. After you have finished editing the XSD document, saving the changes will cause the changes to be saved back to the `types` definitions in the WSDL document. You can close the XSD document without saving changes; in this case, the `types` definitions in the WSDL document will not be modified.

Messages (WSDL 1.1): When a message or its sub-component is selected, the properties of that message or sub-component are displayed in the [Details entry helper](#), where they can be edited. Additionally, you can do the following via the context menu:

- With a message selected in the Overview entry helper, you can add a message part to that message or delete the message, as well as add a new message.
- With a message part selected in the Overview entry helper, you can add another message part to that message or delete the selected message part.
- The Synchronize command highlights the selected message or message part in the relevant portType box.

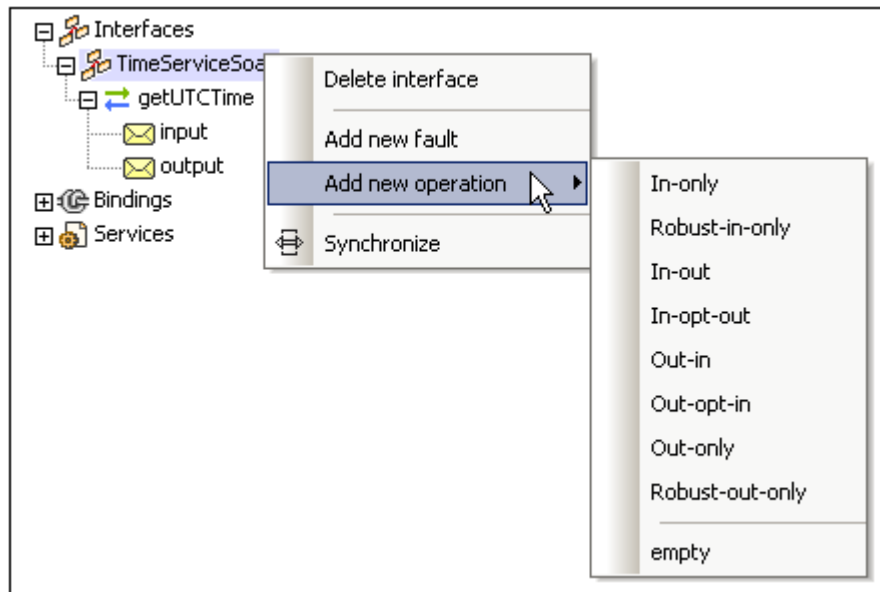
PortTypes (WSDL 1.1): For portTypes, the following functionality is available via context menus.



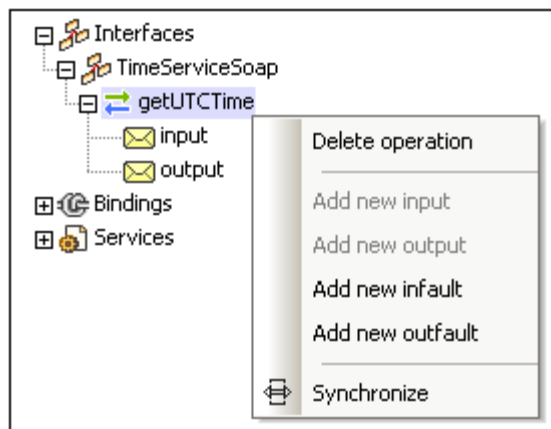
- With the item *PortTypes* selected, a portType can be added.
- With a portType selected, portTypes can be added, the selected portType can be deleted, and operations can be added to the selected portType.
- With an operation selected, additional operations can be appended, the selected operation can be deleted, and elements (input, output, or fault) can be added to the selected operation.
- With a message element (input, output, or fault) selected, additional messages can be added and the selected message can be deleted.
- The Synchronize command highlights the selected portType, operation, or message.

Interfaces (WSDL 2.0): Interfaces can be managed using context menus.

- To add an interface, right-click the *Interfaces* item and select the menu command, **Add new interface**.
- Right-clicking an interface pops up a menu (see screenshot below) with commands enabling the selected interface to be deleted, and faults and operations to be added to the definition of the selected interface. The type of operation to be added can be specified in the submenu of the **Add new operation** command. A corresponding binding operation is added to all bindings referencing the interface. In the same way, when an operation is deleted, referencing binding operations are also deleted.



- Right-clicking an operation pops up commands (see screenshot below) enabling the selected operation to be deleted, and elements (such as infault and outfault) to be added to the operation.



- Right-clicking a message element pops up a menu via which you can delete the selected message.
- Clicking the **Synchronize** command highlights the selected interface, operation, or message in the design.

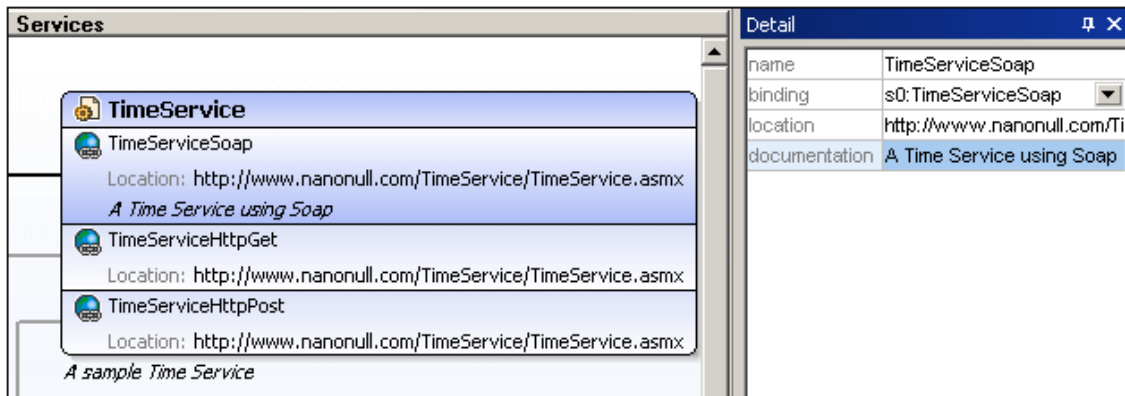
Bindings (WSDL 1.1 and 2.0): With a binding selected, additional bindings can be appended to the already-existing bindings, the selected binding can be deleted, and operations inserted for the selected binding. With an operation or message selected, the same options are available as described for operations and messages in the PortTypes (WSDL 1.1) or Interfaces (WSDL 2.0) category. Clicking the **Synchronize** command highlights the selected binding, operation, or message.

Services (WSDL 1.1 and 2.0): With a service selected, additional services can be added, the selected service can be deleted, and ports can be added for the selected service. Clicking the

Synchronize command highlights the selected service or port.

4.4.3 Details Entry Helper

The Details entry helper displays the properties of the item selected in the Main Window or Overview entry helper (*screenshot below*). These properties can be edited in the Details entry helper.



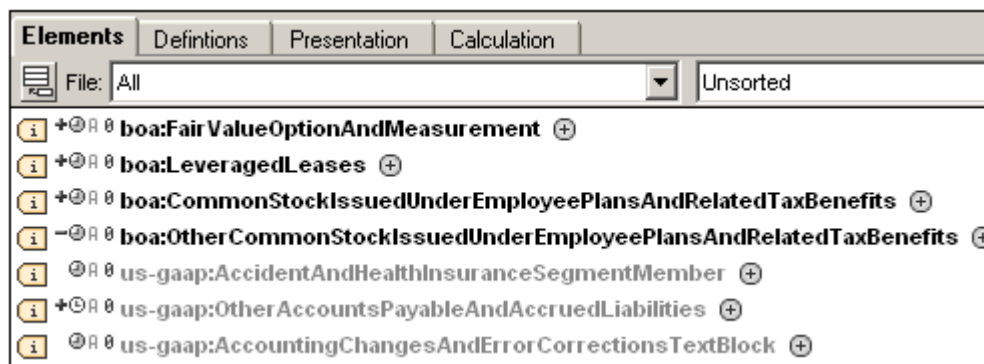
For example, as shown in the screenshot above, if, in the Main Window, the port `TimeServiceSoap` (of the `TimeService` service) is selected, then the properties of `TimeServiceSoap` are displayed in the Details entry helper, where they can be edited. To edit a text field such as for `name` or `description`, double-click in the field and edit the text. For some properties, such as `binding` in the screenshot example above, where a selection can be made from among options, a combo box allows you to select from the available options.

4.5 XBRL View

XBRL View is a graphical interface that enables you to edit XBRL taxonomies. It provides fast validation and error messages, which helps users to develop taxonomies quickly and accurately.

XBRL View consists of the following parts:

- A Main Window having four tabs: [Elements](#), [Definitions](#), [Presentation](#), [Calculation](#) (*screenshot below*). The main features of these tabs are described in the sub-sections of this section.



- Three powerful [entry helpers](#): Overview, Global Elements, Details. These entry helpers enable you to manage taxonomy files and edit the taxonomy in the Main Window.
- A [Messages window](#) which displays messages about the validity of the taxonomy.

This section provides a description of the Main Window and entry helpers of XBRL View and information about how to use them. For more related information, see the sections [XBRL](#) and the description of commands in the [XBRL menu](#).

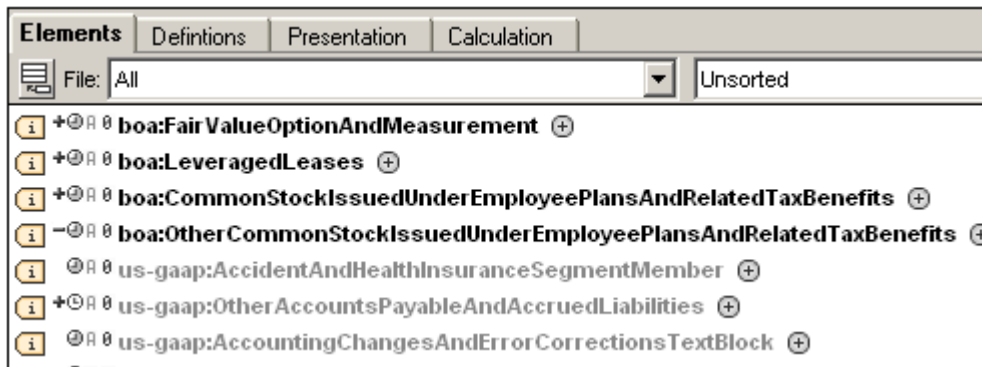
Additional features of XBRL View

Besides the editing features described in this section, the following useful features are available:

- Generate Documentation: which creates detailed documentation files in HTML, Word, and RTF formats.
- [Print of the current view](#) enables a printout of the current view to be taken using XMLSpy's **File | Print** command.

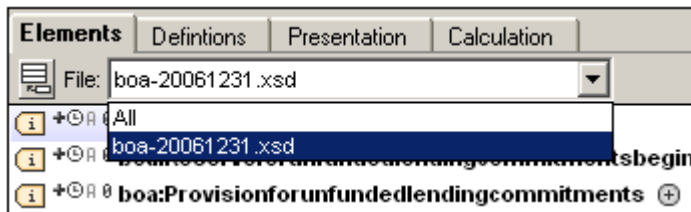
4.5.1 Main Window: Elements Tab

The **Elements** tab of the Main Window (*screenshot below*) displays the concepts of the taxonomy, including, by default, the concepts contained in imported taxonomies. Concepts in the current taxonomy are displayed in black; concepts in imported taxonomies are displayed in gray.



Selecting the taxonomy to display

In the *File* combo box located below the tabs of the Main Window (*screenshot below*), you can select whether concepts from the current taxonomy only, or whether concepts from the current taxonomy plus its imported taxonomies, should be displayed. Select *All* to show the imported taxonomies. Filtering out large imported taxonomies from the display will speed up editing considerably.



Sorting elements

In the *Sort* combo box located below the tabs of the Main Window and to the right (*screenshot below*), you can sort the elements in the Main Window.



The sorting criterion can be one of the following:

- *Unsorted*: Elements are listed in document order.
- *Name*: A namespace prefix is considered part of an element's name.
- *Substitution Group*: There are four substitution groups: *items*; *tuples*; *hypercubeItems*; *dimensionItems*.
- *Sort by Type*: Refers to the Type attribute of the XBRL element.

Finding elements in the Main Window

To find an element in the Main Window press the key combination **Ctrl+F**. This pops up a Find dialog, in which you can enter the string for which you wish to search. The namespace prefixes used in the taxonomy can also be searched.

About concepts (the <element> elements)

Each taxonomy concept is defined in an XML Schema <element> element (*see listing below*). This plain text definition can be seen on switching to the [Text View](#) of the taxonomy document.

(To see the text definitions of an imported taxonomy document, the imported taxonomy document will have to be opened in XMLSpy.)

```
<xs:element id="icui_UnrealizedHoldingLoss"
  name="UnrealizedHoldingLoss"
  substitutionGroup="xbrli:item"
  type="xbrli:monetaryItemType"
  xbrli:balance="credit"
  xbrli:periodType="instant"
  abstract="false"
  nillable="true"/>
```

Each element (or concept) is displayed in the Elements tab with an icon indicating its substitution group (item, tuple, hypercube, or dimension). Additionally, icons indicating the values of the concept's `balance`, `periodType`, `abstract`, and `nillable` attributes are displayed to the left of the concept name (*screenshot below*).



To edit the name of the concept, double-click the name of the concept and edit the name.

Substitution group

The substitution group value of a concept is indicated by the concept's icon:

	xbrli:item
	xbrli:tuple
	xbrldt:hypercubeItem
	xbrldt:dimensionItem

The screenshot below shows an element with a `substitutionGroup` value of `xbrli:item`.



The attributes `balance`, `periodType`, `abstract`, `nillable`

The additional icons to the left of an element's name (*see screenshot above*) indicate the values of the concept's main attributes, respectively, from left to right:

- `xbrli:balance`: a plus icon for a value of `credit`, a minus icon for `debit`
- `xbrli:periodType`: a clock icon with a gray segment between the clock hands for a value of `duration`, white segment for `instant`
- `xs:abstract`: black A icon when `true`, gray when `false`
- `xs:nillable`: black 0 icon when `true`, gray when `false`

Note that the `xbrli:` attributes listed above are from the XBRL schema and the `xs:` attributes are from the XML Schema schema.

In the screenshot above, the plus icon indicates that the value of the `xbrli:balance` attribute is `credit`. The values of the other attributes in the screenshot (`xbrli:periodType`, `xs:abstract`, `xs:nillable`), respectively, are: `duration`, `false`, (i.e. not `abstract`), and `true` (i.e. `nillable`).

The values of these four attributes are displayed in a popup when the cursor is placed over any of the four icons. Clicking any of these icons pops up a list box containing the allowed values for that attribute. You can select one of the allowed choices, and in this way edit the value of

concept attributes quickly.

These attribute values can also be edited in the Details entry helper (*screenshot below*).

Details	
nanonull:OnboardAndOther	
General	
Defined in	file:///S:/Documentation/Public/XBRL/aa-XBRL-MF-example/Taxonomy/nanonull.xsd
Namespace	http://www.altova.com/nanonull
Abstract	☐
Nilable	☒
Balance	credit
Period Type	duration
Subst. Group	xbri:item
Type	xbri:monetaryItemType
Children	
Arcs	0
Label Children	1
Reference Children	0

The attributes substitutionGroup, type

Clicking the arrowhead symbol at the left of the element name expands the display of the element so that the values of the two attributes, `xs:substitutionGroup` and `xs:type`, are displayed graphically (*screenshot below*).

+nanonull:OnboardAndOther	
Substitutiongroup:	xbri:item
Type:	xbri:monetaryItemType
Labels:	http://www.xbrl.org/2003/role/link
en-US	http://www.xbrl.org/2003/role/documentation This is the sale of goods and/or
en-US	http://www.xbrl.org/2003/role/label Onboard and other

In the screenshot above, the value of the `xs:substitutionGroup` and `xs:type` attributes are, respectively: `xbri:item` and `xbri:monetaryItemType`. These values can be edited by selecting alternative values from the dropdown list of the corresponding combo boxes. Both attribute values can also be edited in the Details entry helper (see *screenshot further above*) by double-clicking in the respective value field in the Details entry helper and editing the value via the keyboard.

Label links

To add a label child, right-click an element and select **Add Label Linkrole** from the context menu. This adds a label row (*screenshot below*), in which you can specify the label's properties (language, label type, and value). You can expand and collapse labels by clicking the + and - icons on the right edge of the box.

+us-gaap:TimeDeposits	
Substitutiongroup:	xbri:item
Type:	xbri:monetaryItemType
Labels:	http://www.xbrl.org/2003/role/link
en-US	http://www.xbrl.org/2003/role/label Time Deposits
en-US	http://www.xbrl.org/2003/role/totalLabel Time Deposits, Total

The language and label type properties of each label can be edited by selecting the required property value from in respective combo boxes (*screenshot above*).

Reference links

To add a reference chid, right-click an element and select **Add Reference Linkrole** from the context menu. This adds a reference box, in which the properties of the reference can be edited by clicking in the Reference field and editing the reference URN. You can expand and collapse references by clicking the + and - icons on the right edge of the box.

4.5.2 Main Window: Definitions, Presentation, Calculation Tabs

The Main Window contains tabs for each of the three relationships between concepts:

- Definition relationships, shown in the Definitions tab
- Presentation relationships, shown in the Presentations tab
- Calculation relationships, shown in the Calculations tab

Each tab (Definition, Presentation, and Calculation) displays the taxonomy relationships of that kind (*screenshot below*) and allows you to edit the relationships graphically. A relationship between two concepts (whether a definition, presentation, or calculation relationship) is created by building an arc from one concept to another concept. This *from-to* arc is indicated in the graphical display as a curved arrow. The relationship between the two concepts (in the direction *from-to*) is specified and is known as its arcrole. The arcrole of an arc is shown in the Details entry helper when the element at the *to*-end of a relationship is selected, and, in the case of definition relationships, in an Arcrole column in the Definitions tab (*see screenshot below*).

The way the relationship arcs are displayed is the same for all kind of relationships (definition, presentation, and calculation). In this section, each tab is considered separately, with the basic description of how relationships are displayed being in the section on the Definitions tab. Additional or specific information about presentation and calculation arcs are in the respective sections. For more detailed information, see the sections, [Creating Relationships: Part 1](#) and [Creating Relationships: Part 2](#).

Definitions tab

The **Definitions tab** shows all the definitions of the taxonomy. These definitions are specified in definition arcs contained in definition relationships (.xml) file/s. In the Definitions tab of XBRL View, the structure resulting from the set of definition arcs is displayed in an expandable/collapsible tree form (*screenshot below*).

Editing in the graphical view

The following editing possibilities are available:

- Elements can be dragged from the Global Elements entry helper and dropped in a *to* relationship to an element in the tree.
- An arcrole can be created or edited for an element by selecting the required arcrole (in the *Arcrole* column). The relationship defined in the arcrole expresses a *from-to* relationship, with the selected element occurring at the *to*-endpoint of the relationship.
- Elements can be dragged to alternative locations in the tree. This is a quick way to change the value of the `order` attribute.
- The properties of an element can be edited by clicking one of its property symbols and editing it, or by expanding its property box and then editing properties inside the property box.

Editing in Details entry helper

When an element is selected in the Main Window, the properties of its definition arc are displayed in the Details entry helper in the Arc section (see screenshot below). The values of these properties can be edited by double-clicking in a value field and entering the required value or, if available, by selecting a value in the dropdown list of its combo box.

Details	
company:GenericsSegment	
General	
Defined in	file:///C:/workarea/XBRL/UseCases/UseCase
Namespace	http://www.SampleCompany.com/Company
Abstract	☐
Nilable	☒
Balance	
Period Type	duration
Subst. Group	xbri:item
Type	xbri:monetaryItemType
Arc	
Order	2.0
Priority	0
TargetRole	
Arc Role	http://xbrl.org/int/dim/arcrole/domain-memb
Use	optional
Weight	
Children	
Arcs	0
Label Children	1
Reference Children	0

Additionally, properties of the arc of the selected element are displayed in Arc section of the Details entry helper. The arc will be a definition arc, presentation arc, or calculation arc according to what tab is currently active. The arcrole can be entered in the *Arcrole* field.

Label and reference relationships are listed under the *Children* heading. These relationships can be edited in the Elements tab.

4.5.3 Entry Helpers in XBRL View

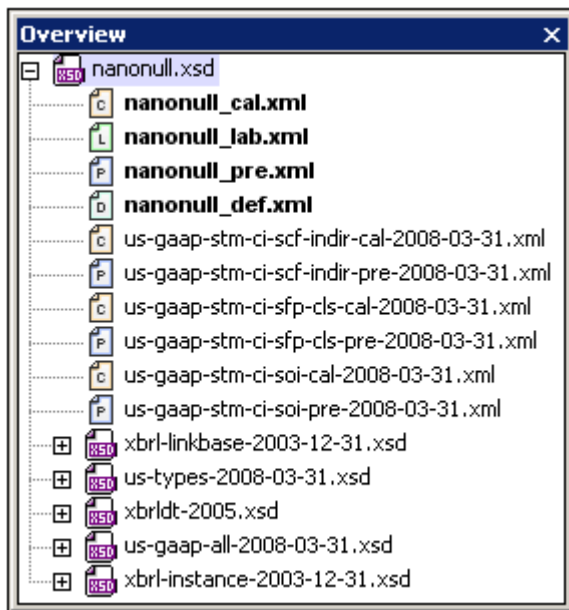
XBRL View features the following entry helpers:

- An [Overview](#) entry helper
- A [Global Elements](#) entry helper
- A [Details](#) entry helper

Validation information is displayed in the [Messages window](#).

Overview entry helper

The Overview entry helper displays the [taxonomy files](#) in a tree structure (*screenshot below*).

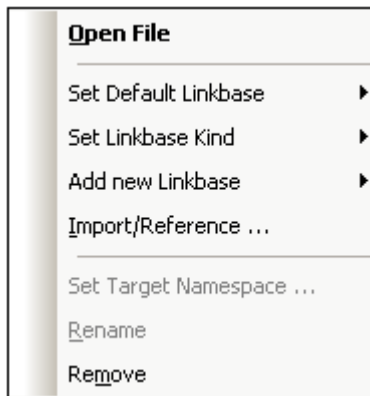


The tree is organized as follows:

- The main concepts file (an XML Schema document) is shown at the root of the tree and is the currently active file.
- The relationship linkbase files (XML files) have a colored file icon with a character corresponding to the initial character of the relationship kind.
 - indicates a definitions linkbase;
 - indicates a labels linkbase;
 - indicates a calculations linkbase;
 - indicates a presentation linkbase; and
 - indicates a reference linkbase.
- Imported schemas are listed on lower levels of the hierarchy. All XML Schema (.xsd) files are indicated with an XSD icon.

This information displayed in the Overview entry helper is obtained from the `/schema/annotation/appinfo` element of the main concepts file. See [Taxonomy Files](#) for more information about this element.

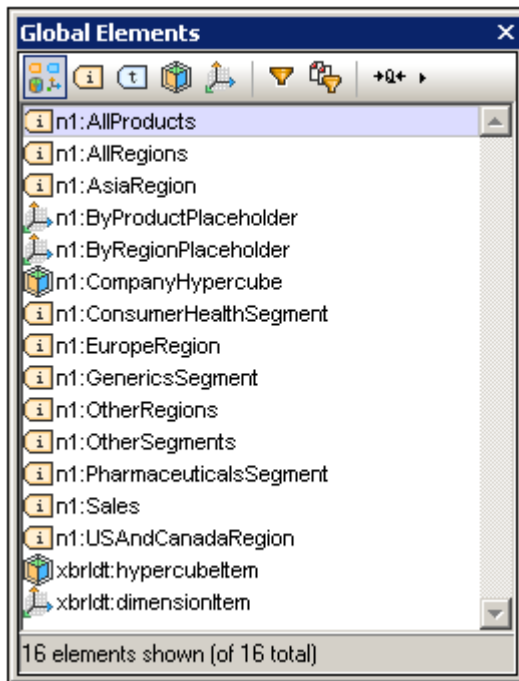
Right-clicking a file in the Overview entry helper pops out a context menu (*screenshot below*), in which the following commands are available:



- **Open File:** opens the selected XML Schema or XML file in XMLSpy.
- **Set Default Linkbase:** If there are multiple files of a single relationship kind (for example, presentation relationships), then one of these can be set as the default linkbase. Newly created relationships will be saved in the default linkbase of its particular relationship kind. The default linkbase of each relationship kind is displayed in bold.
- **Set Linkbase Kind:** Specifies the relationship kind of the selected linkbase. Select the required relationship kind from the submenu that rolls out. The **All** relationship kind specifies that the selected linkbase will be used for all relationship kinds.
- **Add New Linkbase:** Creates a new linkbase file to contain relationships of one of the five kinds (definition, presentation, calculation, label, resource). The added file can be renamed by right-clicking it and selecting the **Rename** command in the context menu. A new `linkbaseRef` element is added to the concepts file; this element references the newly added linkbase.
- **Import/Reference:** Imports a standard taxonomy or creates a reference to an existing XML Schema or linkbase. If you select the standard taxonomy option, a window containing a list of standard taxonomies pops up. Select the taxonomy you wish to import; see [Importing a Taxonomy](#) for details. If you create a reference to an XML schema or linkbase, a new `linkbaseRef` element containing the reference is added to the concepts file. Clicking either the XML Schema or linkbase option pops up a dialog in which you can browse for the required file.
- **Set Target Namespace:** Sets the target namespace and declares this namespace to be in scope on the `xs:schema` element (that is, for the entire taxonomy). See the description of the command [XBRL | Set Target Namespace](#).
- **Rename:** Enables the selected file to be renamed.
- **Remove:** Removes the selected file from the Overview and its `linkbaseRef` element from the concepts file.

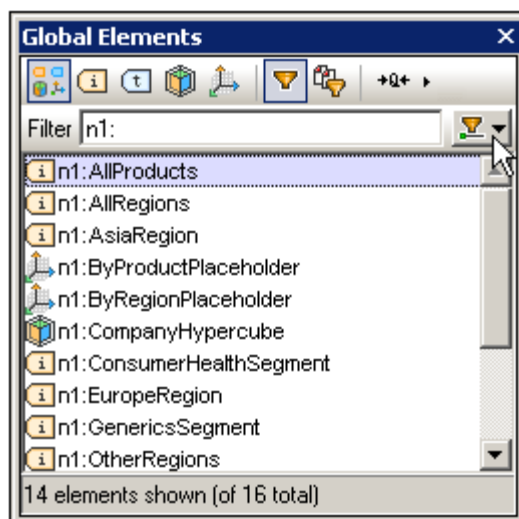
Global Elements entry helper

The Global Elements entry helper (*screenshot below*) displays all the items, tuples, hypercubes, and dimensions present in a taxonomy document. Elements can be dragged from the Global Elements entry helper into the main window.



The following functionality is available in the Global Elements entry helper:

- Filtering by type:** The display of each element type (item, tuple, hypercube, and dimension) can be toggled on and off by clicking its icon in the toolbar of the Global Elements entry helper. Clicking the *Show All Elements* icon  displays all elements. The selected filter is highlighted (Show All, *in the screenshot above*).
- Filtering by text:** Clicking the *Text Filter* icon  displays the Text Filter bar (see *screenshot below*).



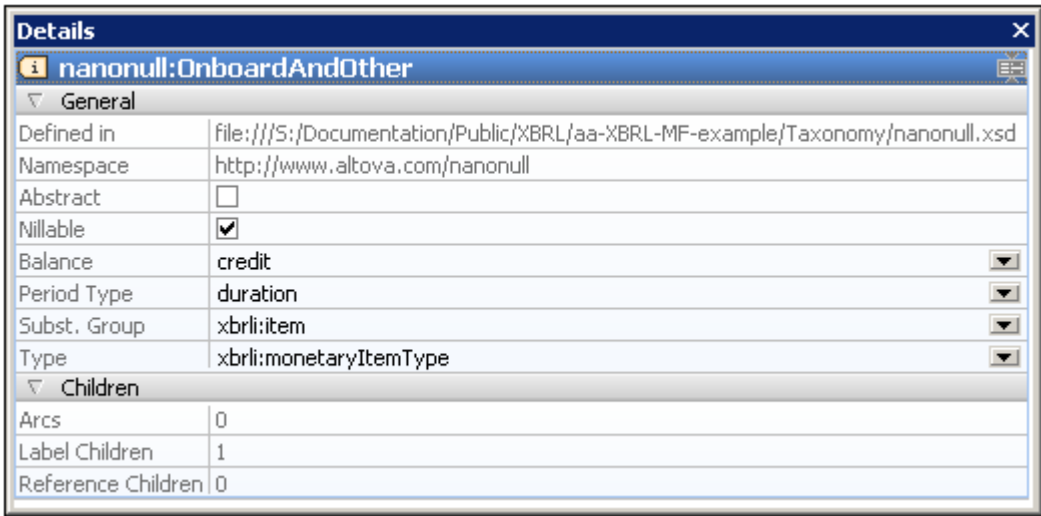
After the Text Filter has been selected, in the combo box at the right-hand side of the Text Filter bar (*under the arrow cursor in the screenshot above*), choose a condition with which to filter. The available options are *Contains*, *Does not contain*, and *Starts with*. In the screenshot above, the *Starts with* condition has been selected. Next, enter the text for the filter in the Text Filter box. The displayed elements will be filtered accordingly. In the screenshot above, the list is filtered by names that begin with `n1`.

Note that: (i) the filters work on the names of the entries as text strings, and (ii) the names of entries can be changed with the Format icon (see below) and are sensitive to the changes in name formats.

- **Filtering by sources:** The required sources can be selected in a popup, and only the elements in the selected sources will be displayed.
- **Format:** There are three format options for the way names of elements are displayed: *Short qualified names*, *Expanded qualified names*, and *Labels*. The short qualified name uses the prefixes assigned to the respective namespaces; expanded qualified name (icon is ) uses the whole namespace; and label uses the labels associated with elements. Note that the format selection affects the *Text Filter* option, in that the filter is applied to elements as listed according to the currently selected *Format* option.
- **Drag-and-drop functionality:** Elements in the Global Elements entry helper can be dragged and dropped into relationships in any of the relationships views of the Main Window (Definitions, Presentations, Calculations)

Details entry helper

When an element is selected in the Main Window, the Details entry helper (*screenshot below*) displays its properties.



Details	
nanonull:OnboardAndOther	
General	
Defined in	file:///S:/Documentation/Public/XBRL/aa-XBRL-MF-example/Taxonomy/nanonull.xsd
Namespace	http://www.altova.com/nanonull
Abstract	☐
Nilable	☒
Balance	credit
Period Type	duration
Subst. Group	xbri:item
Type	xbri:monetaryItemType
Children	
Arcs	0
Label Children	1
Reference Children	0

The properties of some elements can be edited in the Details entry helper: *Abstract*, *Nilable*, *Balance*, *Period Type*, *Substitution Group*, and *Type*.

4.6 Authentic View

Authentic View is enabled by clicking the Authentic tab of the active document. If no SPS has been assigned to the XML document, you are prompted to assign one. You can assign an SPS at any time via the **Authentic | Assign a Stylevision Stylesheet** command.

This section provides:

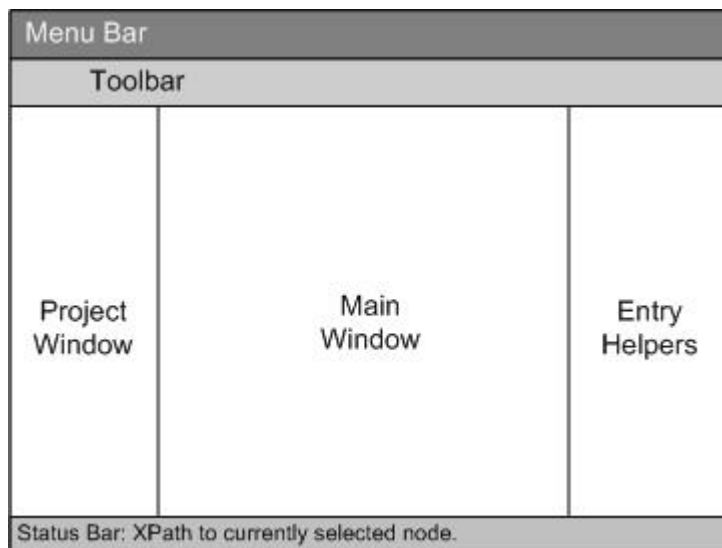
- An overview of the interface
- A description of the toolbar icons specific to Authentic View
- A description of viewing modes available in the main Authentic View window
- A description of the Entry Helpers and how they are to be used
- A description of the context menus available at various points in the Authentic View of the XML document

Additional sources of Authentic View information are:

- An Authentic View Tutorial, which shows you how to use the Authentic View interface. This tutorial is available in the documentation of the Altova XMLSpy and Altova Authentic Desktop products (see the Tutorials section), as well as [online](#).
- For a detailed description of Authentic View menu commands, see the User Reference section of your product documentation.

4.6.1 Overview of the GUI

Authentic View has a menu bar and toolbar running across the top of the window, and three areas that cover the rest of the interface: the Project Window, Main Window, and Entry Helpers Window. These areas are shown below.



Menu bar

The menus available in the menu bar are described in detail in the User Reference section of your product documentation.

Toolbar

The symbols and icons displayed in the toolbar are described in the section, [Authentic View](#)

[toolbar icons](#).

Project window

You can group XML, XSL, XML schema, and Entity files together in a project. To create and modify the list of project files, use the commands in the **Project** menu (described in the User Reference section of your product documentation). The list of project files is displayed in the Project window. A file in the Project window can be accessed by double-clicking it.

Main window

This is the window in which the XML document is displayed and edited. It is described in the section, [Authentic View main window](#).

Entry helpers

There are three entry helper windows in this area: Elements, Attributes, and Entities. What entries appear in these windows (Elements and Attributes Entry Helpers) are context-sensitive, i.e. it depends on where in the document the cursor is. You can enter an element or entity into the document by double-clicking its entry helper. The value of an attribute is entered into the value field of that attribute in the Attributes Entry Helper. See the section [Authentic View Entry Helpers](#) for details.

Status Bar

The Status Bar displays the XPath to the currently selected node.

Context menus

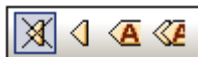
These are the menus that appear when you right-click in the Main Window. The available commands are context-sensitive editing commands, i.e. they allow you to manipulate structure and content relevant to the selected node. Such manipulations include inserting, appending, or deleting a node, adding entities, or cutting and pasting content.

4.6.2 Authentic View Toolbar Icons

Icons in the Authentic View toolbar are command shortcuts. Some icons will already be familiar to you from other Windows applications or Altova products, others might be new to you. This section describes icons unique to Authentic View. In the description below, related icons are grouped together.

Show/hide XML markup

In Authentic View, the tags for all, some, or none of the XML elements or attributes can be displayed, either with their names (large markup) or without names (small markup). The four markup icons appear in the toolbar, and the corresponding commands are available in the **Authentic** menu.



Hide markup. All XML tags are hidden except those which have been collapsed. Double-clicking on a collapsed tag (which is the usual way to expand it) in Hide markup mode will cause the node's content to be displayed and the tags to be hidden.



Show small markup. XML element/attribute tags are shown without names.



Show large markup. XML element/attribute tags are shown with names.



Show mixed markup. In the StyleVision Power Stylesheet, each XML element or attribute can be specified to display (as either large or small markup), or not to display at all. This is called mixed markup mode since some elements can be specified to be displayed with markup and some without markup. In mixed markup mode, therefore, the Authentic View user sees a customized markup. Note, however, that this customization is created by the person who has designed the StyleVision Power Stylesheet. It cannot be defined by the Authentic View user.

Editing dynamic table structures

Rows in a **dynamic SPS table** are repetitions of a data structure. Each row represents an occurrence of a single element. Each row, therefore, has the same XML substructure as the next.

The dynamic table editing commands manipulate the rows of a dynamic SPS table. That is, you can modify the number and order of the element occurrences. You cannot, however, edit the columns of a dynamic SPS table, since this would entail changing the substructure of individual element occurrences.

The icons for dynamic table editing commands appear in the toolbar, and are also available in the **Authentic** menu.



Append row to table



Insert row in table



Duplicate current table row (i.e. cell contents are duplicated)



Move current row up by one row



Move current row down by one row



Delete the current row

Please note: These commands apply only to **dynamic SPS tables**. They should not be used inside static SPS tables. The various types of tables used in Authentic View are described in the [Using tables in Authentic View](#) section of this documentation.

Creating and editing XML tables

You can insert your own tables should you want to present your data as a table. Such tables are inserted as XML tables. You can modify the structure of an XML table, and format the table. The icons for creating and editing XML tables are available in the toolbar, and are shown below. They are described in the section [XML table editing icons](#).



The commands corresponding to these icons are **not available as menu items**. Note also that

for you to be able to use XML tables, this function must be enabled and suitably configured in the StyleVision Power Stylesheet.

A detailed description of the types of tables used in Authentic View and of how XML tables are to be created and edited is given in [Using tables in Authentic View](#).

Text formatting icons

Text in Authentic View is formatted by applying to it an XML element or attribute that has the required formatting. If such formatting has been defined, the designer of the StyleVision Power Stylesheet can provide icons in the Authentic View toolbar to apply the formatting. To apply text formatting using a text formatting icon, highlight the text you want to format, and click the appropriate icon.

DB Row Navigation icons



The arrow icons are, from left to right, Go to First Record in the DB; Go to Previous Record; Open Go to Record # dialog; Go to Next Record; and Go to Last Record.



This icon opens the Edit Database Query dialog in which you can enter a query. Authentic View displays the queried record/s.

XML database editing

The **Select New Row with XML Data for Editing** command enables you to select a new row from the relevant table in an XML DB, such as IBM DB2. This row appears in Authentic View, can be edited there, and then saved back to the DB.

Portable XML Form (PXF) toolbar buttons

The following PXF toolbar buttons are available in the Authentic View of XMLSpy and Authentic Desktop:



Clicking the individual buttons generates HTML, RTF, PDF, and/or DocX output.

These buttons are enabled when a PXF file is opened in Authentic View. Individual buttons are enabled if the PXF file was configured to contain the XSLT stylesheet for that specific output format. For example, if the PXF file was configured to contain the XSLT stylesheets for HTML and RTF, then only the toolbar buttons for HTML and RTF output will be enabled while those for PDF and DocX (Word 2007+) output will be disabled.

4.6.3 Authentic View Main Window

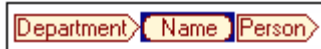
There are four viewing modes in Authentic View: Large Markup; Small Markup; Mixed Markup; and Hide All Markup. These modes enable you to view the document with varying levels of markup information. To switch between modes, use the commands in the **Authentic** menu or the icons in the toolbar (see the previous section, [Authentic View toolbar icons](#)).

Large markup

This shows the start and end tags of elements and attributes with the element/attribute names in the tags:



The element `Name` in the figure above is **expanded**, i.e. the start and end tags, as well as the content of the element, are shown. An element/attribute can be **contracted** by double-clicking either its start or end tag. To expand the contracted element/attribute, double-click the contracted tag.

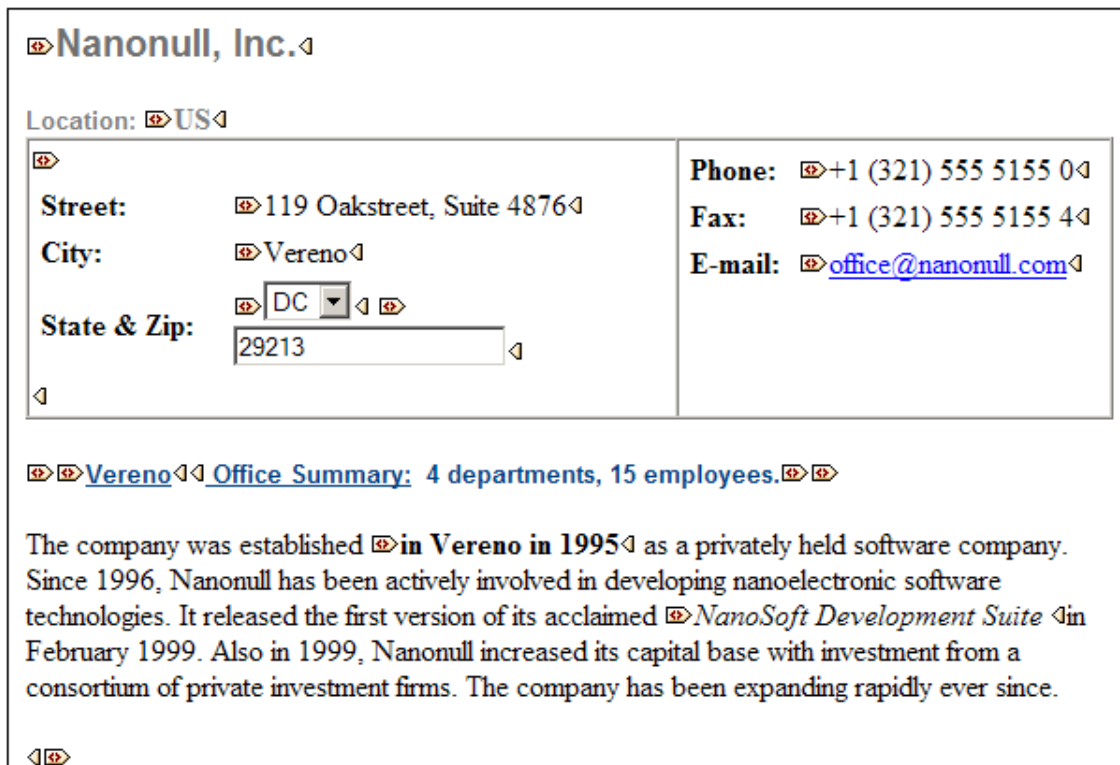


In large markup, attributes are recognized by the equals-to symbol in the start and end tags of the attribute:



Small markup

This shows the start and end tags of elements/attributes without names:



Notice that start tags have a symbol inside it while end tags are empty. Also, element tags have an angular-brackets symbol while attribute tags have an equals sign as its symbol (see screenshot below).



To collapse or expand an element/attribute, double-click the appropriate tag. The example below shows a collapsed element (highlighted in blue). Notice the shape of the tag of the collapsed element and that of the start tag of the expanded element to its left.



Mixed markup

Mixed markup shows a customized level of markup. The person who has designed the StyleVision Power Stylesheet can specify either large markup, small markup, or no markup for individual elements/attributes in the document. The Authentic View user sees this customized markup in mixed markup viewing mode.

Hide all markup

All XML markup is hidden. Since the formatting seen in Authentic View is the formatting of the printed document, this viewing mode is a WYSIWYG view of the document.

Content display

In Authentic View, content is displayed in two ways:

- Plain text. You type in the text, and this text becomes the content of the element or the value of the attribute.



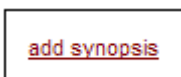
- Data-entry devices. The display contains either an input field (text box), a multiline input field, combo box, check box, or radio button. In the case of input fields and multiline input fields, the text you enter in the field becomes the XML content of the element or the value of the attribute.



In the case of the other data-entry devices, your selection produces a corresponding XML value, which is specified in the StyleVision Power Stylesheet. Thus, in a combo box, a selection of, say, "approved" (which would be available in the dropdown list of the combo box) could map to an XML value of "1", or to "approved", or anything else; while "not approved" could map to "0", or "not approved", or anything else.

Optional nodes

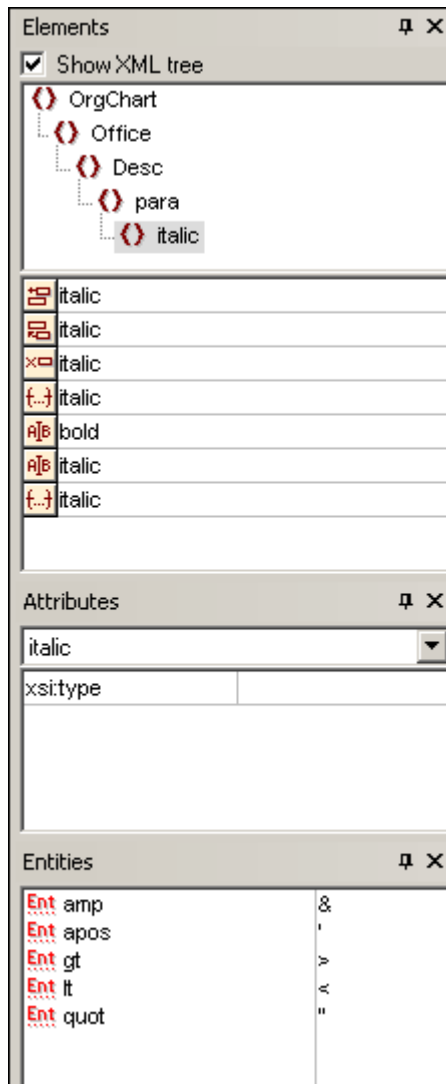
When an element or attribute is **optional** (according to the referenced schema), a prompt of type `add [element/attribute]` is displayed:



Clicking the prompt adds the element, and places the cursor for data entry. If there are multiple optional nodes, the prompt `add. . .` is displayed. Clicking the prompt displays a menu of the optional nodes.

4.6.4 Authentic View Entry Helpers

There are three entry helpers in Authentic View: for Elements, Attributes, and Entities. They are displayed as windows down the right side of the Authentic View interface (*see screenshot below*).



The Elements and Attributes Entry Helpers are context-sensitive, i.e. what appears in the entry helper depends on where the cursor is in the document. The entities displayed in the Entities Entry Helper are not context-sensitive; all entities allowed for the document are displayed no matter where the cursor is.

Each of the entry helpers is described separately below.

Elements Entry Helper

The Elements Entry Helper consists of two parts:

- The upper part, containing an XML tree that can be toggled on and off using the **Show XML tree** check box. The XML tree shows the ancestors up to the document's root element for the current element. When you click on an element in the XML tree,

elements corresponding to that element (as described in the next item in this list) appear in the lower part of the Elements Entry Helper.

- The lower part, containing a list of the nodes that can be inserted within, before, and after; removed; applied to or cleared from the selected element or text range in Authentic View. What you can do with an element listed in the Entry Helper is indicated by the icon to the left of the element name in the Entry Helper. The icons that occur in the Elements Entry Helper are listed below, together with an explanation of what they mean.

To use node from the Entry Helper, click its icon.



Insert After Element

The element in the Entry Helper is inserted after the selected element. Note that it is appended at the correct hierarchical level. For example, if your cursor is inside a `//sect1/para` element, and you append a `sect1` element, then the new `sect1` element will be appended not as a following sibling of `//sect1/para` but as a following sibling of the `sect1` element that is the parent of that `para` element.



Insert Before Element

The element in the Entry Helper is inserted before the selected element. Note that, just as with the Append After Element command, the element is inserted at the correct hierarchical level.



Remove Element

Removes the element and its content.



Insert Element

An element from the Entry Helper can also be inserted within an element. When the cursor is placed within an element, then the allowed child elements of that element can be inserted. Note that allowed child elements can be part of an elements-only content model as well as a mixed content model (text plus child elements).

An allowed child element can be inserted either when a text range is selected or when the cursor is placed as an insertion point within the text.

- When a text range is selected and an element inserted, the text range becomes the content of the inserted element.
- When an element is inserted at an insertion point, the element is inserted at that point.

After an element has been inserted, it can be cleared by clicking either of the two Clear Element icons that appear (in the Elements Entry Helper) for these inline elements. Which of the two icons appears depends on whether you select a text range or place the cursor in the text as an insertion point (see below).



Apply Element

If you select an element in your document (by clicking either its start or end tag in the Show large markup view) and that element can be replaced by another element (for example, in a mixed content element such as `para`, an *italic* element can be replaced by the **bold** element), this icon indicates that the element in the Entry Helper can be applied to the selected (original) element. The **Apply Element** command can also be applied to a text range within an

element of mixed content; the text range will be created as content of the applied element.

- If the applied element has a **child element with the same name** as a child of the original element and an instance of this child element exists in the original element, then the child element of the original is retained in the new element's content.
- If the applied element has **no child element with the same name** as that of an instantiated child of the original element, then the instantiated child of the original element is appended as a sibling of any child element or elements that the new element may have.
- If the applied element has a **child element for which no equivalent exists** in the original element's content model, then this child element is not created directly but Authentic View offers you the option of inserting it.

If a text range is selected rather than an element, applying an element to the selection will create the applied element at that location with the selected text range as its content. Applying an element when the cursor is an insertion point is not allowed.



Clear Element (when range selected)

This icon appears when text within an element of mixed content is selected. Clicking the icon clears the element from around the selected text range.



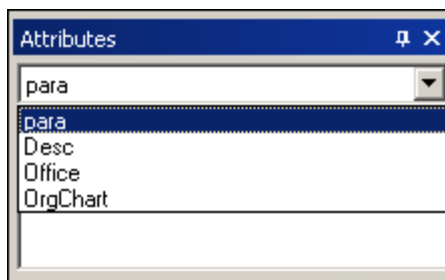
Clear Element (when insertion point selected)

This icon appears when the cursor is placed within an element that is a child of a mixed-content element. Clicking the icon clears the inline element.

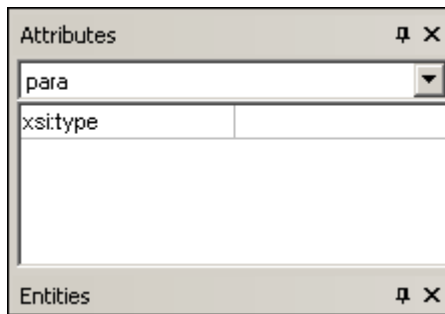
Attributes Entry Helper

The Attributes Entry Helper consists of a drop-down combo box and a list of attributes. The element that you have selected (you can click the start or end tag, or place the cursor anywhere in the element content to select it) appears in the combo box.

The Attributes Entry Helper shown in the figures below has a `para` element in the combo box. Clicking the arrow in the combo box drops down a list of all the `para` element's **ancestors up to the document's root element**, which in this case is `OrgChart`.



Below the combo box, a list of valid attributes for that element is displayed, in this case for `para`. If an attribute is mandatory on a given element, then it appears in bold. (In the example below, there are no mandatory attributes except the built-in attribute `xsi:type`.)



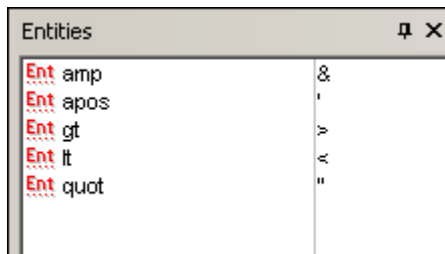
To enter a value for an attribute, click in the value field of the attribute and enter the value. This creates the attribute and its value in the XML document.

In the case of the `xsi:nil` attribute, which appears in the Attributes Entry Helper when a nillable element has been selected, the value of the `xsi:nil` attribute can only be entered by selecting one of the allowed values (`true` or `false`) from the dropdown list for the attribute's value.

The `xsi:type` attribute can be changed by clicking in the value field of the attribute and then selecting, from the dropdown list that appears, one of the listed values. The listed values are the available abstract types defined in the XML Schema on which the Authentic View document is based.

Entities Entry Helper

The Entities Entry Helper allows you to insert an entity in your document. Entities can be used to insert special characters or text fragments that occur often in a document (such as the name of a company). To insert an entity, place the cursor at the point in the text where you want to have the entity inserted, then double-click the entity in the Entities Entry Helper.



Note: An internal entity is one that has its value defined within the DTD. An external entity is one that has its value contained in an external source, e.g. another XML file. Both internal and external entities are listed in the Entities Entry Helper. When you insert an entity, whether internal or external, the entity—not its value—is inserted into the XML text. If the entity is an internal entity, Authentic View displays **the value of the entity**. If the entity is an external entity, Authentic View displays the entity—and not its value. This means, for example, that an XML file that is an external entity will be shown in the Authentic View display as an entity; its content does not replace the entity in the Authentic View display.

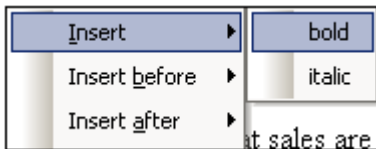
You can also **define your own entities** in Authentic View and these will also be displayed in the entry helper: see [Define Entities](#) in the Editing in Authentic View section.

4.6.5 Authentic View Context Menus

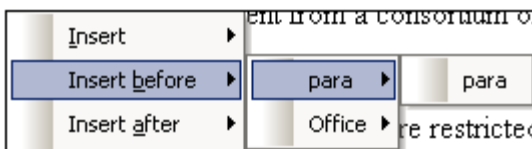
Right-clicking on some selected document content or node pops up a context menu with commands relevant to the selection or cursor location.

Inserting elements

The figure below shows the **Insert** submenu, which is a list of all elements that can be inserted at that current cursor location. The **Insert Before** submenu lists all elements that can be inserted before the current element. The **Insert After** submenu lists all elements that can be inserted after the current element. In the figure below, the current element is the `para` element. The **bold** and **italic** elements can be inserted within the current `para` element.



As can be seen below, the `para` and `Office` elements can be inserted before the current `para` element.



The node insertion, replacement (**Apply**), and markup removal (**Clear**) commands that are available in the context menu are also available in the [Authentic View entry helpers](#) and are fully described in that section.

Insert entity

Positioning the cursor over the Insert Entity command rolls out a submenu containing a list of all declared entities. Clicking an entity inserts it at the selection. See [Define Entities](#) for a description of how to define entities for the document.

Insert CDATA Section

This command is enabled when the cursor is placed within text. Clicking it inserts a CDATA section at the cursor insertion point. The CDATA section is delimited by start and end tags; to see these tags you should switch on large or small markup. Within CDATA sections, XML markup and parsing is ignored. XML markup characters (the ampersand, apostrophe, greater than, less than, and quote characters) are not treated as markup, but as literals. So CDATA sections are useful for text such as program code listings, which have XML markup characters.

Remove node

Positioning the mouse cursor over the **Remove** command pops up a menu list consisting of the selected node and all its removable ancestors (those that would not invalidate the document) up to the document element. Click the element to be removed. This is a quick way to delete an element or any removable ancestor. Note that clicking an ancestor element will remove all its descendants, including the selected element.

Clear

The **Clear** command clears the element markup from around the selection. If the entire node is

selected, then the element markup is cleared for the entire node. If a text segment is selected, then the element markup is cleared from around that text segment only.

Apply

The **Apply** command applies a selected element to your selection in the main Window. For more details, see [Authentic View entry helpers](#).

Copy, Cut, Paste

These are the standard Windows commands. Note, however, that the **Paste** command pastes copied text either as XML or as Text, depending on what the designer of the stylesheet has specified for the SPS as a whole. For information about how the **Copy as XML** and **Copy as Text** commands work, see the description of the **Paste As** command immediately below.

Paste As

The **Paste As** command offers the option of pasting as XML or as text an Authentic View XML fragment (which was copied to the clipboard). If the copied fragment is pasted as XML it is pasted together with its XML markup. If it is pasted as text, then only the text content of the copied fragment is pasted (not the XML markup, if any). The following situations are possible:

- An **entire node together with its markup tags** is highlighted in Authentic View and copied to the clipboard. (i) The node can be pasted as XML to any location where this node may validly be placed. It will not be pasted to an invalid location. (ii) If the node is pasted as text, then only the node's *text content* will be pasted (not the markup); the text content can be pasted to any location in the XML document where text may be pasted.
- A **text fragment** is highlighted in Authentic View and copied to the clipboard. (i) If this fragment is pasted as XML, then the XML markup tags of the text—even though these were not explicitly copied with the text fragment—will be pasted along with the text, but only if the XML node is valid at the location where the fragment is pasted. (ii) If the fragment is pasted as text, then it can be pasted to any location in the XML document where text may be pasted.

Note: Text will be copied to nodes where text is allowed, so it is up to you to ensure that the copied text does not invalidate the document. The copied text should therefore be:

- (i) lexically valid in the new location (for example, non-numeric characters in a numeric node would be invalid), and
- (ii) not otherwise invalidate the node (for example, four digits in a node that accepts only three-digit numbers would invalidate the node).

If the pasted text does in any way invalidate the document, this will be indicated by the text being displayed in red.

Delete

The **Delete** command removes the selected node and its contents. A node is considered to be selected for this purpose by placing the cursor within the node or by clicking either the start or end tag of the node.

4.7 Browser View

Browser View is typically used to view:

- XML files that have an associated XSLT file. When you switch to Browser View, the XML file is transformed on the fly using the associated XSLT stylesheet and the result is displayed directly in Browser View.
- HTML files which are either created directly as HTML or created via an XSLT transformation of an XML file.

To view XML and HTML files in Browser View, click the **Browser** tab.

Note about Microsoft Internet Explorer and XSLT

Browser View requires Microsoft's Internet Explorer 5.0 or later. If you wish to use Browser View for viewing XML files transformed by an XSLT stylesheet, we strongly recommend Internet Explorer 6.0 or later, which uses MSXML 3.0, an XML parser that fully supports the XSLT 1.0 standard. You might also wish to install MSXML 4.0. Please see our [Download Center](#) for more details. (Note that support for XSLT in IE 5 is not 100% compatible with the official XSLT Recommendation. So if you encounter problems in Browser View with IE 5, you should upgrade to IE 6 or later.) You should also check the support for XSLT of your version of Internet Explorer.

Browser View features

The following features are available in Browser View. They can be accessed via the [Browser menu](#), [File menu](#), and [Edit menu](#).

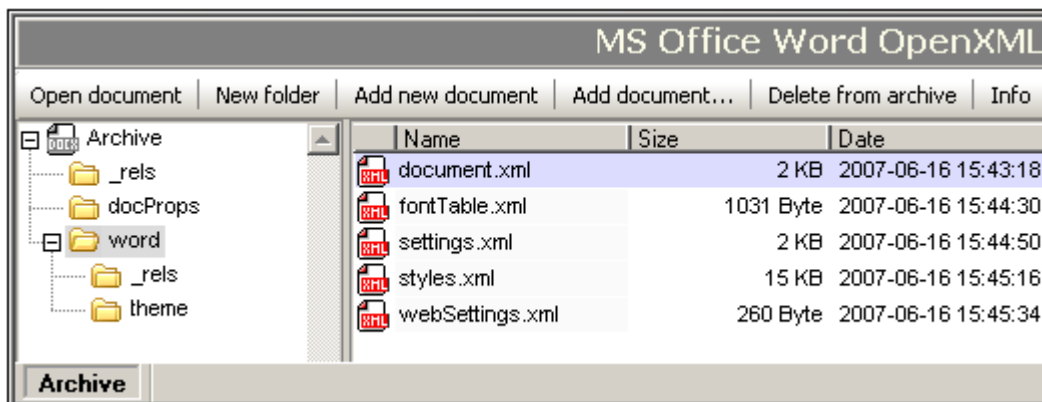
- **Open in separate window:** When Browser View is a separate window, it can be positioned side-by-side with an editing view of the same document. This command is in the **Browser** menu and is a toggle-command that can be used to return a separate Browser View window as a tabbed view. In the [View tab](#) of the Options dialog, you can set whether Browser View should, by default, be a separate window.
- **Forward and Back:** The common browser commands to navigate through pages that were loaded in Browser View. These commands are in the **Browser** menu.
- **Font size:** Can be adjusted via the **Browser** menu command.
- **Stop, Refresh, Print:** More standard browser commands, these can be found in the **Browser** and **File** menus.
- **Find:** Enables searches for text strings, this command is in the **Edit** menu.

4.8 Archive View

An [Office Open XML \(OOXML\)](#) file, [ZIP file](#) (for example, WinZip or WinRAR), or [EPUB file](#) can be opened and edited in Archive View. Not only can OOXML, ZIP, and EPUB archives be structurally modified in Archive View, but individual files in the archive can be opened from Archive View, edited in one of XMLSpy's editing views, and then saved directly back to the archive.

Archive files and Archive View

When an archive file (OOXML, ZIP, or EPUB file) is [created or opened in XMLSpy](#), it is opened in Archive View (*screenshot below*). Multiple archive files can be open at a time, with each archive file being in a separate Archive View window. The type of the archive file appears in the top right-hand corner of Archive View. In the screenshot below, the type of the archive file is MS Office Word Open XML.



Folder View

The Folder View is located on the left-hand side of the Archive View window and displays the folder structure of the zipped archive. On each level, folders are listed alphabetically. To view the sub-folders of a folder, click the plus symbol to the left of the folder. If a folder does not have a plus symbol to the left of it, then it has no sub-folder. To view the document files (hereafter called documents) contained in a folder, select the folder; the files will be displayed in the Main Window. In the screenshot above, the documents displayed in the Main Window are in the `word` folder, which also has two sub-folders: `_rels` and `theme`.

Main Window

The Main Window lists the documents in the folder that is selected in Folder View. Documents are displayed in alphabetical order, each with its respective uncompressed size and the date and time of last modification. To open a Document from Archive View, double-click it. The document opens in a separate XMLSpy window.

Command buttons

The command buttons are located along the top of the Archive View window.

- **Open document:** Enabled when a document in the Main Window is selected. Clicking it opens the selected document. A document can also be opened by double-clicking the document listing in the Main Window.
- **New folder:** Adds a new folder to the folder that is currently selected in Folder View. The folder must be named immediately upon its being created in Folder View. It is not possible to rename a folder subsequently. The new folder is saved in the archive when

the archive file is saved.

- **Add new document:** Adds a new document to the folder currently selected in Folder View. Clicking this button opens the Create New Document dialog of XMLSpy. The newly created document opens in a separate XMLSpy window. The document must be named immediately upon its being listed in the document listing of the selected folder. The document is saved in the archive only when it is saved in its own editing window or when the archive file is saved.
- **Add document:** Opens a Browse dialog in which you can browse for a document to add. The document is added to the listing in the Main Window of documents currently in the selected folder, and the document is opened in a separate XMLSpy window. For the document to be saved to the archive, it must either be saved in its own window, or the archive file must be saved.
- **Delete from archive:** Deletes the selected document (in Main Window) or selected folder (in Folder View) from the archive. The archive file must be saved in order for the deletion to take effect.
- **Info:** Toggles the Info Window on and off. *See below.*

Info Window

The Info Window is toggled on and off by clicking the Info command button. The Info Window provides general information about the archive file, such as the number of files it contains, its uncompressed and compressed sizes, and the compression ratio.

5 XML

This section describes how to work with XML documents in XMLSpy. It covers the following aspects:

- [How to create, open, and save XML documents](#). In this section, some important XMLSpy settings relating to file creation are also explained.
- XML documents can be edited in [Text View](#), [Grid View](#), and [Authentic View](#). You can select the view that is most useful for you and switch among the views while editing. Each of the views offers different advantages.
- [Entry helpers](#) for XML documents have certain specific features, and these are described.
- How to [process XML documents with XSLT and XQuery](#). Various XMLSpy features related to processing are explained.
- Miscellaneous [other features](#) for working with XML documents are described.

Altova website:  [XML Editor](#)

5.1 Creating, Opening, and Saving XML Documents

When creating, opening, or saving XML documents, the following issues are involved:

- In what view will the XML document open: Text View, Grid View, or Authentic View
- When a new XML document is created, whether a schema (XML Schema or DTD) will be automatically assigned, manually assigned, or not assigned
- If a schema is assigned to the XML document, whether the document will be validated automatically on opening and/or saving

Default view

There are application-wide settings for specifying in what view XML documents (new and existing) should open. These settings are in the Options dialog (**Tools | Options**).

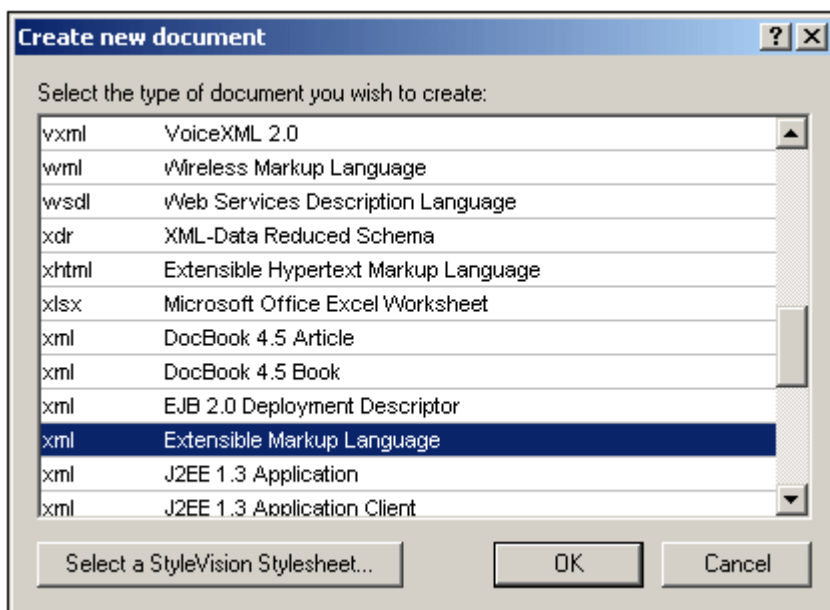
In the **File Types** tab of the Options dialog, select a file type of `.xml` and, in the Default View pane, check the required editing view (Text or Grid). Note that: (i) Schema View and WSDL View can be used only for XML Schema and WSDL documents, respectively; and (ii) Browser View is a display view, not an editing view.

In the **File Types** tab, you can also set XMLSpy as the default editor for the selected file type.

An XML document can be edited in Authentic View if a StyleVision Power Stylesheet (SPS) has been assigned to it. When an XML file with an associated SPS is opened, you can specify that it opens directly in Authentic View. Do this by checking the *Always open in Authentic View* option in the **View** tab of the Options dialog. If this option is not checked, the file will open in the default view specified for `.xml` files in the **File Types** tab (see above).

Assigning schemas

When a new XML file is to be created, select the menu command **File | New**. This pops up the Create New Document dialog (screenshot below).



Notice that there are several options for the XML document type. The option marked *Extensible Markup Language* creates a generic XML document. Each of the other options is associated with a schema, for example the DocBook DTD. If you select one of these options, an XML

document is created that has (i) the corresponding schema automatically assigned to it, and (ii) a skeleton document structure that is valid according to the assigned schema. Note that you can create your own skeleton XML document. If you save it in the `Template` folder of the application folder, your skeleton document will be available for selection in the Create New Document dialog.

If you select the generic Extensible Markup Language document type, you will be prompted for a schema (DTD or XML Schema) to assign to the document. At this point, you can choose to browse for a schema or go ahead and create an XML document with no schema assigned to it.

You can, of course, assign a schema via the **DTD/Schema** menu at any subsequent time during editing.

Automatic validation

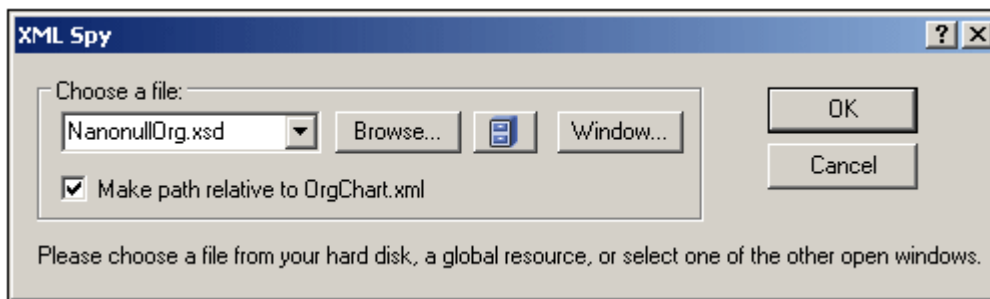
If an existing XML document has a schema assigned to it, then it can be automatically validated on opening and/or saving. The setting for this is in the **File** tab of the Options dialog (**Tools | Options**).

The automatic validation settings in the **File** tab can be combined with a setting in the File Types tab to disable automatic validation for specific file types. Using the settings in the two tabs together enables you to specify automatic validation for specific file types.

5.2 Assigning Schemas and Validating

Altova website:  [XML Validator](#), [XML Validation](#)

A schema (DTD or XML Schema) can be assigned to an XML document [when it is first created](#). A schema can also be assigned, or changed, at any subsequent time using the **Assign DTD** or **Assign Schema** commands in the **DTD/Schema** menu.



The path to the schema file that is inserted in the XML document can be made relative by checking the relative path check box in the dialog.

Global resources for schemas

A global resource is an alias for a file or folder. The target file or folder can be changed within the GUI by changing the active configuration of the global resource (via the menu command **Tools | Active Configuration**). Global resources therefore enable the assigned schema to be switched among multiple schemas, which can be useful for testing. How to use global resources is described in the section [Altova Global Resources](#).

XML Schema plus DTD

One very useful DTD feature that XML Schema does not have is the use of entities. However, if you wish to use entities in your XML-Schema-validated XML document, you can add a `DOCTYPE` declaration to the XML document and include your entity declarations in it.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE OrgChart [
  <! ENTITY name-int "value">
  <! ENTITY name-ext SYSTEM "extfile.xml">
]>
<OrgChart xmlns="http://www.xs.com/org"
  xsi:schemaLocation="http://www.xs.com/org OrgChart.xsd">
  ...
</OrgChart>
```

After declaring the entities in the DTD, they can be used in the XML document. The document will be well-formed and valid. Note, however, that external parsed entities are not supported in Authentic View..

Going to schema definitions

With the XML document open, you can directly open the DTD or XML Schema on which it is based by clicking the **Go to DTD** or **Go to Schema** commands in the **DTD/Schema** menu. Additionally, you can place the cursor within a node in the XML document and go to the schema definition of that node via the **Go to Definition** command in the **DTD/Schema** menu.

Validating and checking well-formedness

To validate and/or check for well-formedness, use the [Validate XML \(F8\)](#) and **Check Well-Formedness (F7)** commands in the XML menu or the corresponding commands in the toolbar. Any error is reported in the Messages window.

5.3 Editing XML in Text View

XMLSpy offers some specialized XML text editing features in addition to the generally available editing features in Text View, which are described in [Text View](#) in the Editing Views section.

- [Commenting text in and out](#)
- [Escaping and unescaping XML characters](#)
- [Inserting file paths](#)
- [Inserting XML fragments via XInclude](#)
- [Copying XPath and XPointer expressions to the clipboard](#)

Commenting text in/out

Text in an XML document can be commented out using the XML start-comment and end-comment delimiters, respectively `<!--` and `-->`. In XMLSpy, these comment delimiters can be easily inserted using the **Edit | Comment In/Out** menu command.

To comment out a block of text, select the text to be commented out and then select the command **Comment In/Out**, either from the **Edit** menu or the context menu that you get on right-clicking the selected text. The commented text will be grayed out (*see screenshot below*).



```

<Department>
  <Name>Administration</Name>
  <Person>
  <Person>
  <Person>
  <!--<Person>
    <First
    <Last></Last>
    <PhoneExt></PhoneExt>
    <EMail></EMail>
    <LeaveTotal></LeaveTotal>
    <LeaveUsed></LeaveUsed>
    <LeaveLeft></LeaveLeft>
  </Person>-->
</Department>
  
```

To uncomment a commented block of text, select the commented block **excluding** the comment delimiters, and select the command **Comment In/Out**, either from the **Edit** menu or the context menu that you get on right-clicking the selected text. The comment delimiters will be removed and the text will no longer be grayed out.

Note about empty lines

In XML documents, empty lines are discarded when you change views or save the document. If you wish to retain empty lines, enclose them in comment delimiters.

Escaping and unescaping XML characters

The five XML special characters (*listed below*) can be escaped and unescaped with the corresponding entity references (*listed below*) by highlighting a block of text and selecting the context menu command **Escape XML Characters** or **Unescape XML Characters**. The XML special characters in that block of text will then be escaped or unescaped according to the command selected.

```

<    &lt;
>    &gt;
&    &amp;
  
```

' '
" "

For example:

<a> can be escaped with the **Escape XML Characters** command to
 <a>
<a> can be unescaped with the **Unescape XML Characters** command
 to <a>

Inserting file paths

The [Edit | Insert File Path](#) command enables you to browse for the file in question and insert its file path at the selected location in the XML document being edited. This command enables you to quickly and accurately enter a file path. See the [command description](#) for more details.

Inserting XML fragments via XInclude

The [Edit | Insert XInclude](#) enables you, via XInclude, to insert the contents of an entire XML document, or a fragment of one, in the XML document being edited. This command enables you to quickly and accurately enter a file path. See the [command description](#) for more details.

Copying XPath and XPointer expressions to the clipboard

The XPath and XPointer expressions of the selected node can be copied to the clipboard using the [Edit | Copy XPath](#) and [Edit | Copy XPointer](#) commands, respectively. This enables you to obtain the correct XPath and XPointer expressions for a given node. The copied expressions can then be inserted at the required location, for example, an XPath expression in an XSLT stylesheet and an XPointer expression in the `href` attribute of an `xinclude` element. For more detailed descriptions of the commands, see their descriptions in the User Reference section.

5.4 Editing XML in Grid View

[Grid View](#) shows the hierarchical structure of **XML documents** through a set of nested containers that can be expanded and collapsed. This provides a clear picture of the document's structure. In Grid View, both structure and contents can be easily manipulated.

XML

Company

= xmlns	http://my-company.com/namespace
= xmlns:xsi	http://www.w3.org/2001/XMLSchema-instance
= xsi:schema...	http://my-company.com/namespace AddressLast.xsd

Address

= xsi:type	US-Address
Name	US dependency
Street	Noble Ave.
City	Dallas
Zip	04812
State	Texas

Person (3)

	= Manager	= Degree	= Programmer	First	Last
1	false	MA	true	Alfred	Alford
2	true	Ph.D	false	Colin	Colman
3	true	BA	false	Fred	Smith

In the screenshot above, notice that the document is displayed as a hierarchy in a grid form. When a node can contain content, it is divided into two fields: for name and for content. Node names are displayed in bold face and node content in normal face.

Display as table

If there are several instances of a repeating element, then, in standard Grid View, each complete instance is displayed, one after the other, progressing vertically downward in document order (*screenshot below*).

Person		
First		Fred
Last		Landis
Title		Program Manager
PhoneExt		951
EMail		f.landis@nanonull.com
Shares		2000
LeaveTotal		28
LeaveUsed		10
LeaveLeft		18

Person		
First		Michelle
Last		Butler
Title		Software Engineer
PhoneExt		654
EMail		m.butler@nanonull.com
Shares		1500
LeaveTotal		19
LeaveUsed		9
LeaveLeft		10

Such a structure of multiple instances can also be displayed as a table (*screenshot below*), in which the child nodes form the columns and the multiple instances form the rows.

Person (6)			
	First	Last	Title
1	Fred	Landis	Program Manager
2	Michelle	Butler	Software Engineer
3	Ted	Little	Software Engineer
4	Ann	vWay	Technical Writer
5	Liz	Gardner	Software Engineer
6	Paul	Smith	Software Engineer

Table View offers a unique editing advantage in that whole rows and columns can be manipulated relative to other columns and rows in the table. This enables such operations as sorting data with respect to the value of one common child node. For example, in the screenshot above, the six `Person` elements can be sorted on the basis of their `Last` child elements via a simple GUI operation. Such an operation is much simpler than running an XSLT transformation, which would be the usual way to sort an XML nodeset.

Editing the document structure

In Grid View, the XML document structure can be edited graphically. For example, you can collapse and expand individual segments of the document structure, insert, append and delete nodes, drag-and-drop nodes to different locations, and convert one type of node to another type.

The **XML** menu offers commands to insert, append, and add empty child nodes. For example, you can add an empty child node by selecting an element and then adding an empty child element. You can then enter a name and content for the newly added node by double-clicking in the respective field (name or content field) and entering the required string.

The **Elements and Attributes entry helpers** enable you to insert, append, and add child nodes that are allowed at the selected location. For example, you select a node in the Main Window. The element and attribute nodes that may be validly inserted, appended, or added as a child at this location are listed in the Elements and Attributes entry helpers.

The commands available in the XML menu and entry helpers that are applicable to a selected node are also available in the context menu of that node.

Editing content

To edit content, double-click in the content field and type in the content text. Entities can be inserted via the Entities entry helper.

More about Grid View

For a more detailed description of Grid View, see under [Editing Views](#).

5.5 Editing XML in Authentic View

Authentic View enables a user to edit an XML document as if it were a text document (*screenshot below*). The XML markup and all other non-content text can be hidden from the person editing the document. This can be useful for people who are unfamiliar with XML, enabling them to a valid XML document even while concentrating entirely on the content of the document..

Location: US	
Street: 119 Oakstreet, Suite 4876	Phone: +1 (321) 555 5155 0
City: Vereno	Fax: +1 (321) 555 5155 4
State & Zip: DC 29213	E-mail: office@nanonull.com

Vereno Office Summary: 4 departments, 15 employees.

The company was established **in Vereno in 1995** as a privately held software company. Since 1996, Nanonull has been actively involved in developing nanoelectronic software technologies. It released the first version of its acclaimed *NanoSoft Development Suite* in February 1999. Also in 1999, Nanonull increased its capital base with investment from a consortium of private investment firms. The company has been expanding rapidly ever since.

Due to the fact that nanoelectronic software components are new and that sales are restricted to corporate customers, Nanonull and its product line have not received much media publicity in the company's early years. This has however changed in recent months as trade journals have realized the importance of this revolutionary technology.

The Authentic View of a document is enabled when a StyleVision Power Stylesheet (SPS) is assigned to an XML document. An SPS is based on the same schema source as that on which the XML document is based, and it defines the structure of the XML document. The SPS also defines the layout and formatting of the document in Authentic View. For example, in the document shown in the screenshot above, the following Authentic formatting and editing features are used:

- Paragraph and other block formatting
- Table structures
- Text formatting, such as color and font face
- Combo boxes (see the State and Zip fields) enable the user to select from a group of valid choices, which can be taken from schema enumerations, as has been done in the case above
- Additional information can be calculated from the data in the document and be presented (in the example above, the office summary details have not been entered by the user but calculated from other data in the document)

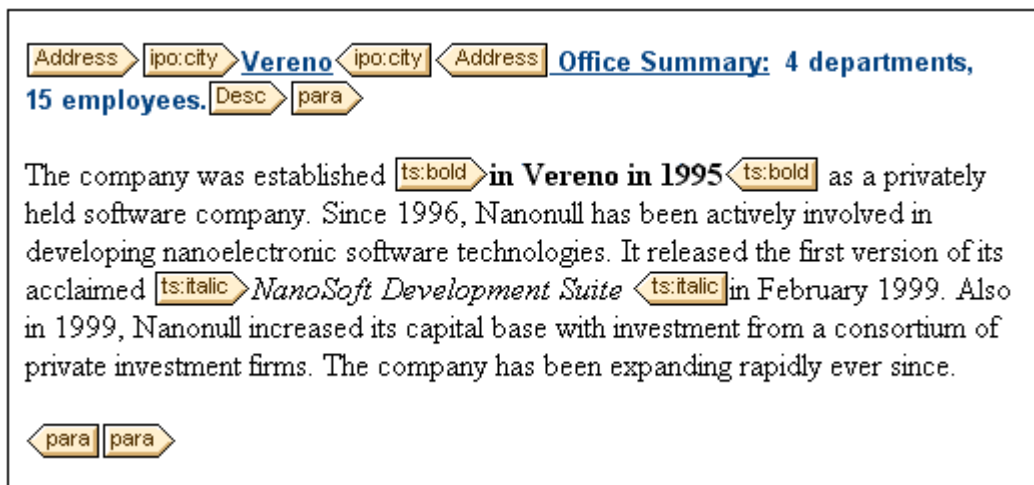
SPSs are created specifically for viewing and editing XML documents in Authentic View and for

generating standard output (such as HTML, PDF, RTF, and Word 2007 documents) from XML. SPSs are created with Altova StyleVision.

Editing document structure

Valid nodes can be added to the document at any time by selecting a location and then adding the required node via the entry helpers (Elements and Attributes) or context menu. The nodes available at any given location are restricted to the nodes that can be validly added as siblings or children at the selected location. For example, when the cursor is located within a paragraph, you can append another paragraph if this is allowed by the schema.

When editing the structure of an XML document in Authentic View, it could be useful to see the markup of the document. Markup can therefore be switched on as tags (*screenshot below*) using the **Authentic | Show Large Markup** command (or the corresponding toolbar icon).



Editing content

Content is created and edited by typing it into the nodes of the document. Entities and CDATA sections can be added via the context menu (entities also via the Entities entry helper).

More about editing in Authentic View

For more details of how to edit in Authentic View, see the Authentic View section.

5.6 Entry Helpers for XML Documents

For XML documents, there are three entry helpers: Elements entry helper; Attributes entry helper; and Entities entry helper. When an element is added via the Elements entry helper, it can be added together with mandatory child elements, mandatory attributes, all child elements, or no child element or attribute, according to the respective settings in the [Editing tab of the Options dialog](#). When empty attributes are added, they are added with quotes.

Note that in the different views, the entry helpers are designed differently, in accordance with the functionality of the respective view.

Elements entry helper

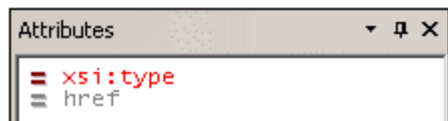
The following points should be noted:

- *Text View*: Elements are inserted at the cursor insertion point. Unused elements are displayed in red, used elements in gray. Mandatory elements are indicated with an exclamation mark "!" before the name of the element.
- *Grid View*: Elements can be appended after, inserted before, or added as a child of the selected element. There are therefore three tabs, each displaying the elements that may be added. Unused elements are displayed in black, used elements in gray.
- *Authentic View*: Elements can be inserted before, after, or within the selected element. Additionally, there is a document tree that shows the location of the currently selected element in the document's tree structure. For more details of how to edit in Authentic View, see the Authentic View section.

Attributes entry helper

The following points should be noted:

- *Text View*: When the cursor is placed inside the start tag of an element and after a space, the attributes declared for that element become visible. Unused attributes are displayed in red, used attributes in gray. Mandatory attributes are indicated with an exclamation mark "!" before the name of the attribute.



- To insert an attribute, double-click the required attribute. The attribute is inserted at the cursor point together with an equals-to sign and quotes to delimit the attribute value. The cursor is placed between the quotes, so you can start typing in the attribute value directly.
- *Grid View*: When an element is selected, the attributes that can be added as a child are listed in the Add Child tab of then entry helper. When an attribute is selected, the available attributes are listed in the Append (after) and Insert (before) tabs. Unused attributes are displayed in black, used attributes in gray.
- *Authentic View*: When an element is selected, the attributes declared for that element become visible. Enter he value of the attribute in the entry helper.

Entities entry helper

Any parsed or unparsed entity that is declared inline (within the XML document) or in an external DTD, is displayed in the Entities entry helper. In all three views (Text, Grid, and Authentic), an entity is inserted at the cursor insertion point by double-clicking it. In Grid View, entities are displayed in the Append, Insert, and Add Child tabs.

Note that if you add an internal entity, you will need to save and reopen your document before the entity appears in the Entities entry helper.

5.7 Processing with XSLT and XQuery

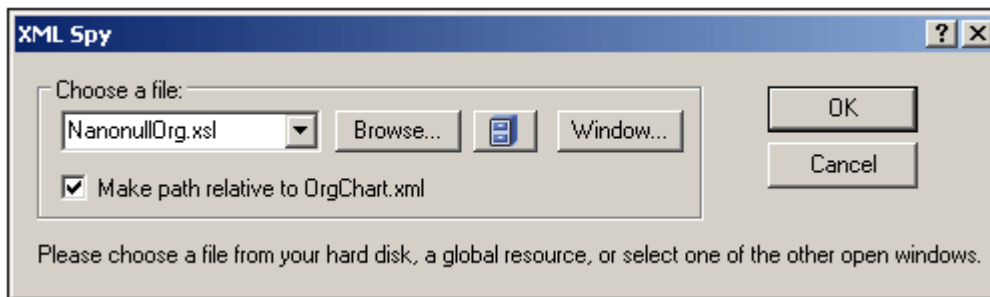
XML documents can be processed with XSLT or XQuery documents to produce output documents. XMLSpy has built-in XSLT 1.0, XSLT 2.0, and XQuery 1.0 processors. The following processing-related features are available in the GUI:

- [Assigning XSLT stylesheets](#)
- [Go to XSLT](#)
- [XSLT parameters and XQuery variables](#)
- [XSLT transformations](#)
- [XQuery executions](#)
- [Automating XML tasks with AltovaXML](#)

Assigning XSLT stylesheets

You can assign an XSLT stylesheet to an XML document via the **XSL/XQuery | Assign XSL** command (browse for the file in the dialog (*screenshot below*) that pops up). The assignment is entered in the XML document as a processing instruction (PI) having the standard XSLT target defined by the W3C: `xml-stylesheet`. This assignment is used when an XSLT transformation is invoked (**XSL/XQuery | XSL Transformation**).

Additionally, an XSLT-for-FO stylesheet can be assigned with the **XSL/XQuery | Assign XSL: FO** command (browse for the file in the dialog (*screenshot below*) that pops up). The assignment is entered in the XML document as a processing instruction (PI) having the Altova-defined target: `altova_xslfo`. This assignment is used when an XSLT-for-FO transformation is invoked (**XSL/XQuery | XS:FO Transformation**).



You can also select a global resource to specify the XSLT file. A global resource is an alias for a file or folder. The target file or folder can be changed within the GUI by changing the active configuration of the global resource (via the menu command **Tools | Active Configuration**). Global resources therefore enable the assigned XSLT file to be switched from one to another, which can be useful for testing. How to use global resources is described in the section [Altova Global Resources](#).

If a previous assignment using either of these PI targets exists, then you are asked whether you wish to overwrite the existing assignment.

Go to XSLT

The **XSL/XQuery | Go to XSL** command opens the XSLT file that has been assigned to the XML document.

XSLT parameters and XQuery variables

XSLT parameters and XQuery variables can be defined, edited, and deleted in the dialog that

appears on clicking the command **XSL/XQuery | XSLT Parameters / XQuery Variables**. The parameter/variable values defined here are used for all XSLT transformations and XQuery executions in XMLSpy. However, these values will not be passed to external engines such as MSXML. For the details of how to use this feature, see the [User Reference section](#).

XSLT transformations

Two types of XSLT transformation are available:

- Standard XSLT transformation (**XSL/XQuery | XSL Transformation**): The output of the transformation is displayed in a new window or, if specified in the stylesheet, is saved to a file location. The engine used for the transformation is specified in the [XSL tab](#) of the Options dialog ([Tools | Options](#)).
- XSL-for-FO transformation (**XSL/XQuery | XSL-FO Transformation**): The XML document is transformed to PDF in a two-step process. In the first step, the XML document is transformed to an FO document using the XSLT processor specified in the [XSL tab](#) of the Options dialog ([Tools | Options](#)); note that you can also select (at the bottom of the tab) the XSLT engine that comes with some FO processors such as FOP. In the second step, the FO document is processed by the FO processor specified in the [XSL tab](#) of the Options dialog ([Tools | Options](#)) to produce PDF output.

Note: An FO document (which is a particular type of XML document) can be transformed to PDF by clicking the XSL:FO transformation command. When the source document is an FO document, the second step of the two-step process for this command is executed directly.

XQuery executions

An XQuery document can be executed on the active XML document by clicking the command **XSL/XQuery | XQuery Execution**. You are prompted for the XQuery file, and the result document is displayed in a new window in the GUI.

Automating XML tasks with AltovaXML

AltovaXML is a free application which contains Altova's XML Validator, XSLT 1.0, XSLT 2.0, and XQuery 1.0 engines. It can be used from the command line, via a COM interface, in Java programs, and in .NET applications to validate XML documents, transform XML documents using XSLT 1.0 and 2.0 stylesheets, and execute XQuery documents.

XSLT transformation tasks can therefore be automated with the use of AltovaXML. For example, you can create a batch file that calls AltovaXML to transform a set of documents. See the [Altova XML documentation](#) for details.

5.8 Charts

When an XML document is open in Text View or Grid View, a chart (pie chart, bar chart, etc) representing selected data in the XML document can be generated in the Charts Window (which is one of the [Output Windows](#)). The chart can then be exported as an image file or as an XSLT or XQuery fragment to the clipboard. The Charts feature is useful for quickly representing selected numeric data in an XML document graphically.

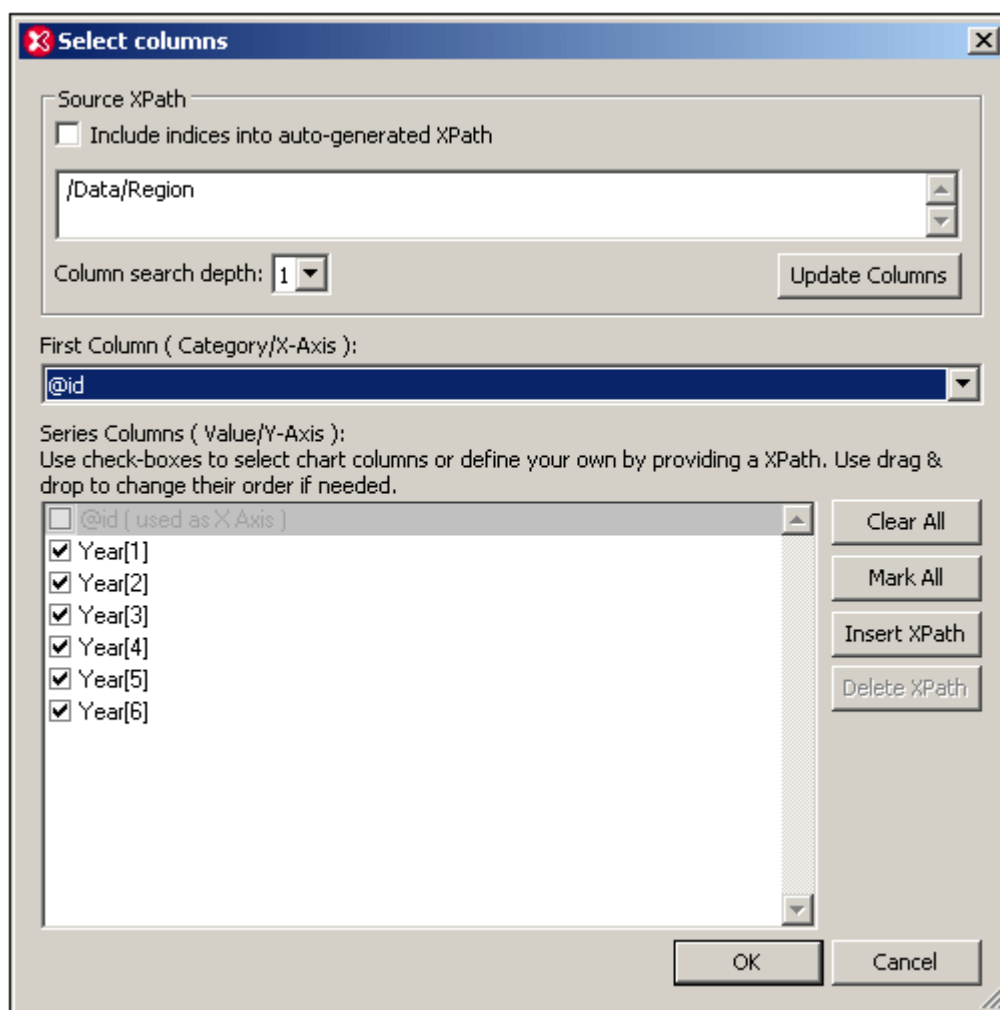
The following chart types are available:

- Pie charts (2D, 3D)
- Bar charts, single bars (2D, 3D)
- Bar charts, grouped bars (2D, 3D)
- Stacked bar charts
- Category line graphs
- Value line graphs
- Area charts and stacked area charts
- Candlestick charts
- Gauge charts (round and bar)
- Overlay charts

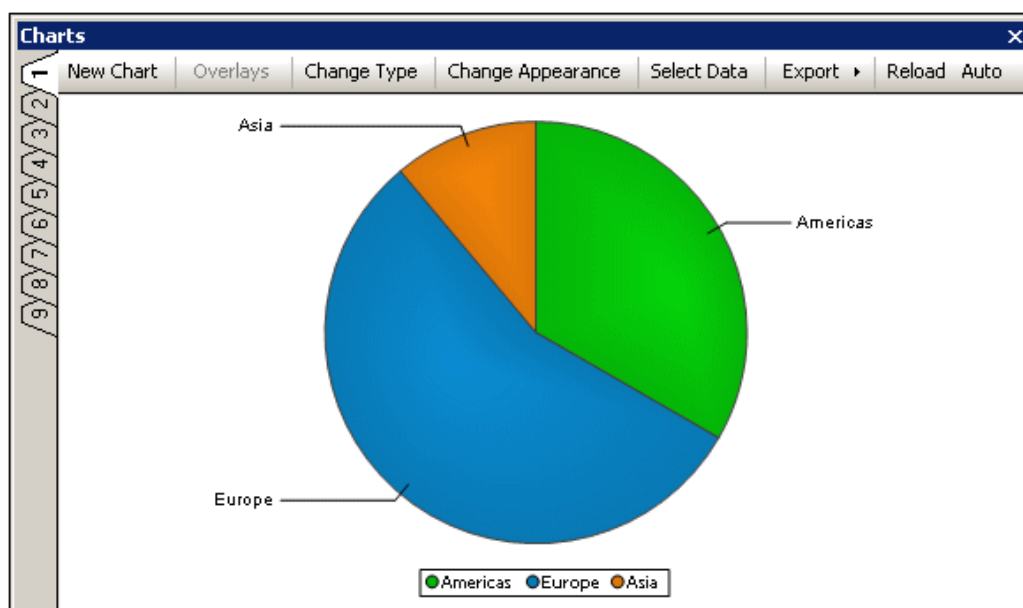
Overview: from creation to export

The steps to create a chart are described broadly below. For a more detailed account, see the subsections of this section.

1. In Text View or Grid View, select the node that you wish to use as the context node for the data selection. You can also select a range of nodes. The implications of the various selection methods are explained in the section [Source XPath](#).
2. Right-click and, from the context menu that appears, select the command **New Chart**. Alternatively, in the Charts output window, click the **New Chart** button. This pops up the Select Columns dialog (*screenshot below*), in which the [X-Axis](#) and [Y-Axis](#) data will be selected and in which the [Source XPath](#) can be modified.



3. On clicking **OK**, the chart is created in the Charts Window (see screenshot below).



4. The chart's data selection and other settings can subsequently be edited. Not only can its Source XPath and column selection be edited, but also its type and appearance. The data selection for the chart's axes can be edited by clicking the *Select Data* button. And the chart's type and appearance can be modified by clicking the **Change Type** button and **Change Appearance** button, respectively.
5. The chart can be exported as an image file or as an XSLT or XQuery fragment to the clipboard.

Other features

The following features help to make usage easier:

- **Multiple tabs:** If you wish to create a new chart without deleting the current chart, then create the new chart in any one of the other tabs marked one to nine (see *screenshot above*). Note that, even when an XML document is closed, charts generated from that document will stay open in their respective tabs in the Charts Window.
- **Auto Reloading:** If the **Auto** button (see *screenshot above*) is toggled on, then the chart will be automatically reloaded every time data in the XML document is modified. Otherwise, the chart will have to be manually updated by clicking the **Reload** button.

Example file

In this section and subsection, explanations about how charts work reference an XML file called *YearlySales.xml*. This file is available in the folder `C:\Documents and Settings\<username>\My Documents\Altova\XMLSpy2012\Examples\Tutorial`.

```
<?xml version="1.0" encoding="UTF-8"?>
<Data xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="YearlySales.xsd">
  <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>
```

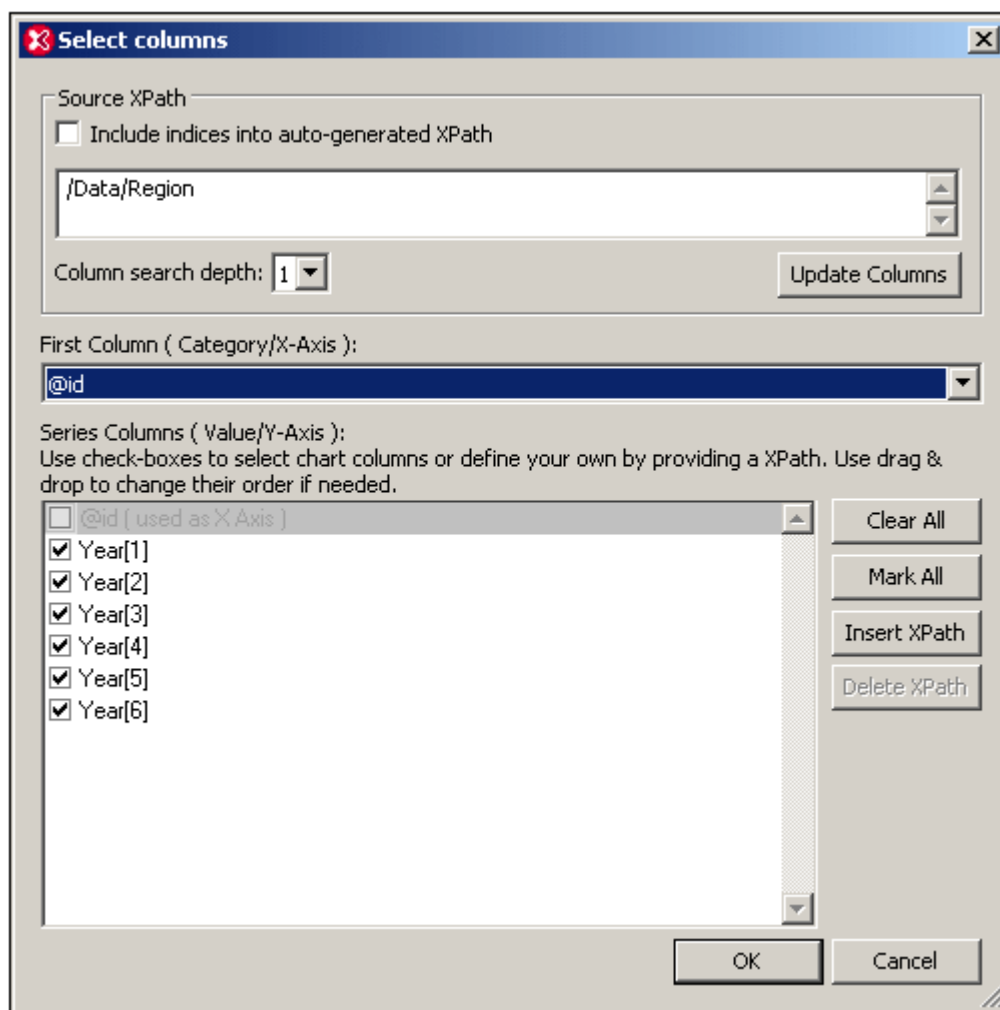
5.8.1 Creating a Chart

The **New Chart** button pops up the Select Columns dialog (*screenshot below*), in which three fundamental data selection parameters for the chart are specified. These parameters (listed below) are used to build up the chart data table.

- **Source XPath:** An XPath expression is automatically entered when the dialog opens. It selects the node in the XML document that was selected when the Select Columns dialog was accessed. It can be edited in the dialog using the keyboard. The *Include Indices* checkbox determines whether predicate filters in the XPath will be used or not

(see [Source XPath](#) for details). Descendant nodes of the node/s selected by the Source XPath will be available for selection as X-Axis and Y-Axis data columns. The Column Search Depth combo box determines how many descendant levels will be searched to return nodes that may be used for X-Axis and Y-Axis data selection. After the Source XPath has been edited, **Update Columns** must be clicked for the change to take effect and for the X-Axis and Y-Axis lists in the dialog to be refreshed.

- **X-Axis:** The selection in this combo box specifies which node will be used as the X-Axis. The sequence returned for this selection will give the labels that occur on the X-Axis. The *Auto-Enumerated* option of the combo box provides numbered labels for the X-Axis. Note that XPath expressions created for the Y-Axis are also available for selection in the X-Axis combo box.
- **Y-Axis:** The entries that are checked in this pane will be the nodes, the numeric values of which will be represented on the numeric Y-Axis. The **Clear All** and **Mark All** buttons deselect all items and select all items in the Y-Axis pane, respectively. The **Insert XPath** button enables a series to be generated that is not available because it is not a descendant of the node the Source XPath returns. The node or XPath expression selected for the X-Axis is not available for Y-Axis selection and is grayed out.



How the chart data table is created

The data that is used for the chart is determined by the selection made in the Select Columns

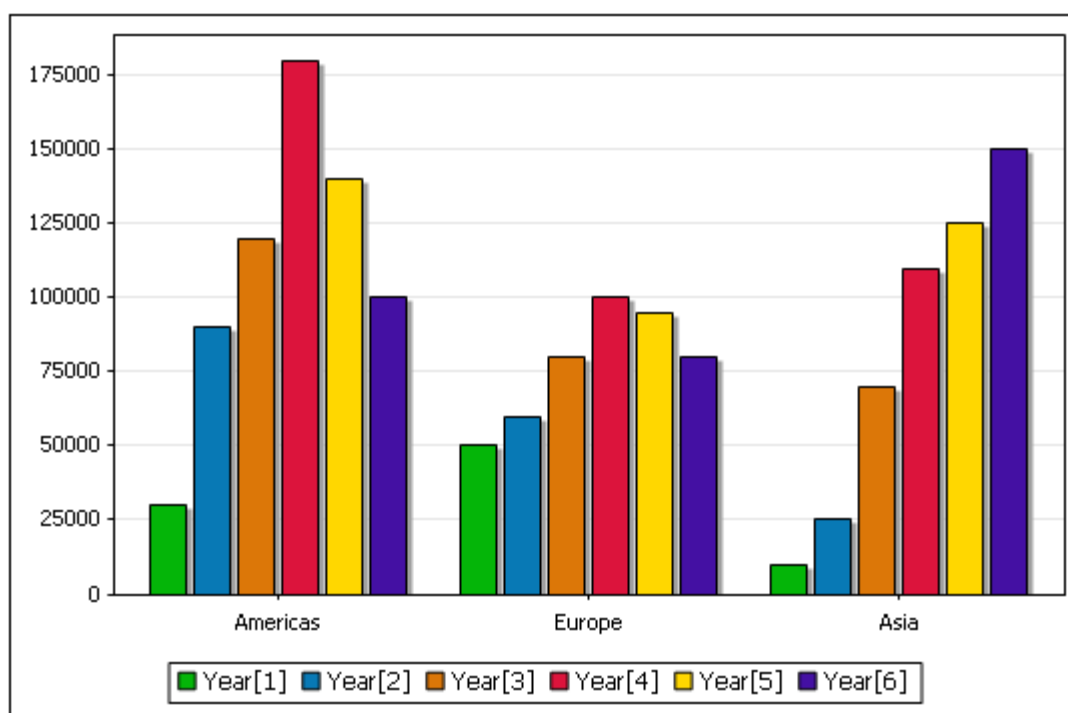
dialog. We will explain how the chart data is selected with the help of an example. Since the XML document (*see screenshot further below*) contains three `Region` elements, the Source XPath `/Data/Region` selects each of them in turn. With each `Region` element as the context node, the columns of data are then generated. For each `Region` element selected because of the Source XPath the following is done:

1. The X-Axis expression generates the first column (by default this column will be the column used for the X-Axis labels).
2. For each series (Y-Axis selections) a column is generated.

The chart data generated for the Select Columns dialog shown above can be visualized as in the following table.

Source XPath	X-Axis	Y-Axis (Series columns)					
Region[1]	@id	Year[1]	Year[2]	Year[3]	Year[4]	Year[5]	Year[6]
Region[2]	@id	Year[1]	Year[2]	Year[3]	Year[4]	Year[5]	Year[6]
Region[3]	@id	Year[1]	Year[2]	Year[3]	Year[4]	Year[5]	Year[6]

A bar chart generated from this data would look something like this:



The following important points should be noted:

- The number of ticks on the X-Axis is determined by the size of the sequence returned by the Source XPath expression (in this case three).
- The nodes returned by the Source XPath will be the context nodes, respectively, for generating two sets of data for each tick on the X-Axis: (i) the X-Axis tick label (made with the X-Axis selection), and (ii) all the series to be plotted for that tick (these series are selected with the Y-Axis selection). The XPath expressions entered for the X-Axis and Y-Axis will be evaluated as XPath expressions in the context of these (Source XPath) nodes.
- The sequence returned by the X-Axis selection will be, respectively, the label for each tick. If there are fewer labels than there are ticks, then some ticks will remain

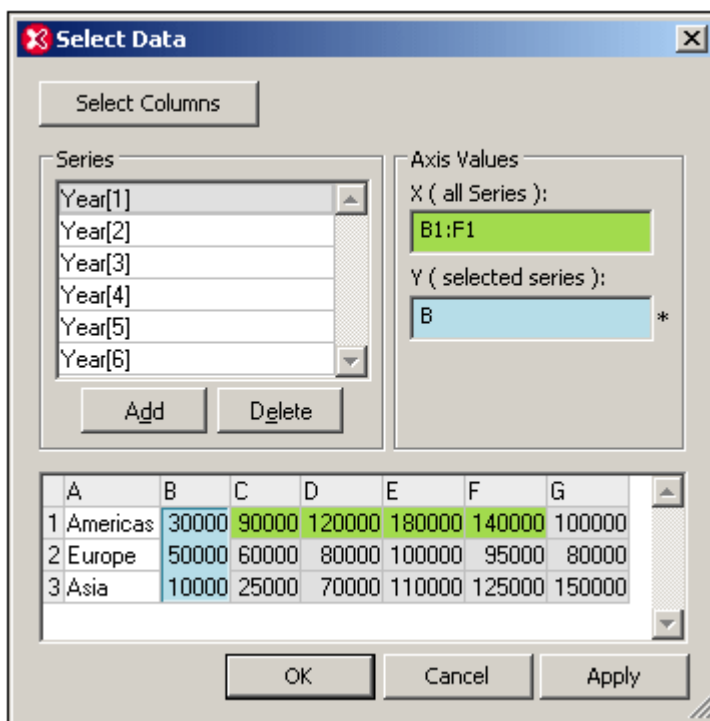
unlabelled.

- Each series (for example `Year[ 1 ]`) is evaluated once for each context node. For some charts, like pie charts or single-bar charts, only one series can be used.
- The legends are obtained from the names of series items.

The XML document used for the example above is given here for reference. It is named `YearlySales.xml` and is available in the folder `C:\Documents and Settings\<username>\My Documents\Altova\XMLSpy2012\Examples\Tutorial`.

```
<?xml version="1.0" encoding="UTF-8"?>
<Data xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance "
      xsi:noNamespaceSchemaLocation="YearlySales.xsd">
  <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>
```

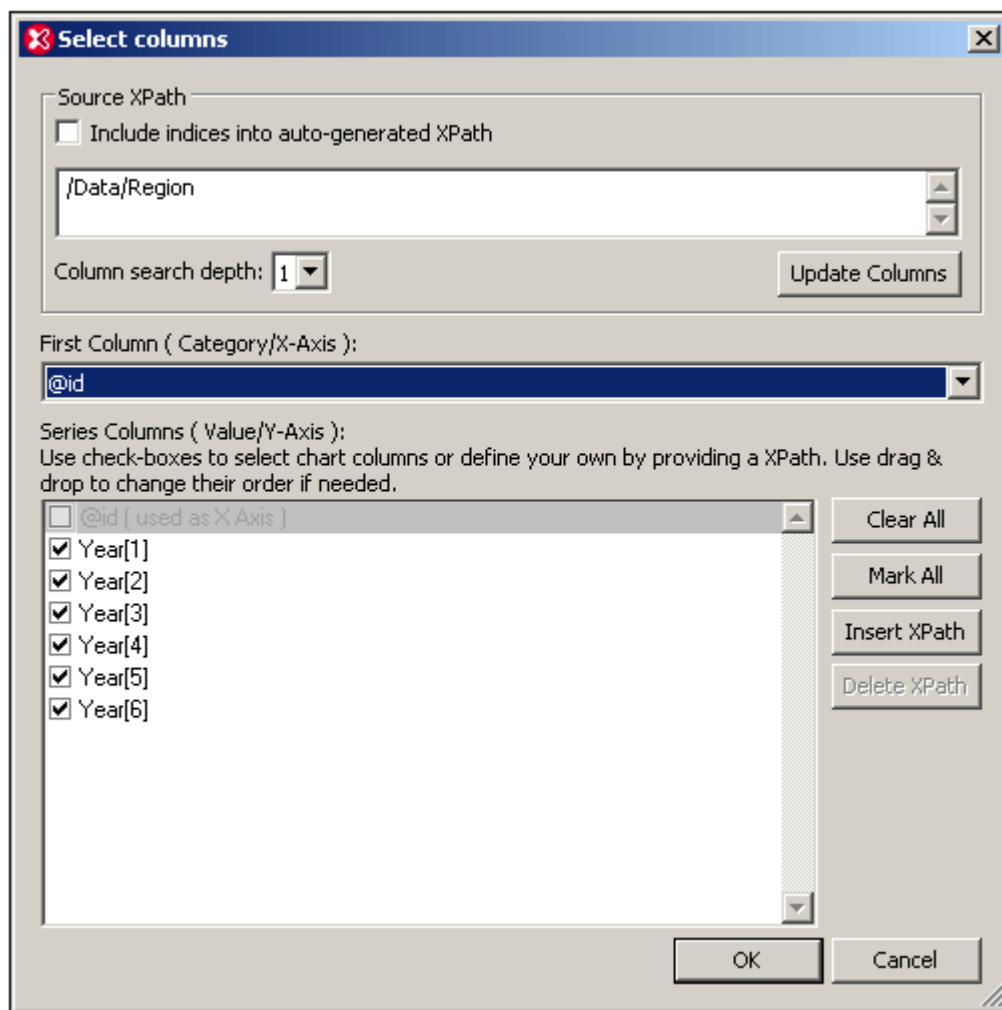
The data selection shown in the Select Columns dialog above can be seen in the table of the Select Data dialog (*screenshot below*). The Select Data dialog is accessed by clicking the **Select Data** button in the Charts output window.



For more details about the individual parameters of the Select Columns dialog, see the individual sections: [Source XPath](#), [X-Axis Selection](#), [Y-Axis Selection](#), and [Chart Data](#).

5.8.2 Source XPath

The Source XPath is specified in the Select Columns dialog. It determines what nodes in the document are available for selection as X-Axis and Y-Axis data. The Column Search Depth combo box determines how many descendant levels will be searched to return nodes that may be used for the X-Axis and Y-Axis data.



After a new Source XPath has been selected, clicking **Update Columns** refreshes the available selections (in the dialog) for the X-Axis and Y-Axis. If predicates are included in the XPath expression (for example, `/Data/Region[1]` uses the `[1]` predicate to select the first `Region` element), then the Include Indices check box must be checked. If you modify the Source XPath expression be sure to click **Update Columns**.

Source XPath from cursor location

To explain what Source XPath is selected when the Select Columns dialog is opened, we'll use the XML document shown below. It is named `YearlySales.xml` and is available in the folder `C:\Documents and Settings\<username>\My Documents\Altova\XMLSpy2012\Examples\Tutorial`.

```
<?xml version="1.0" encoding="UTF-8"?>
<Data xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="YearlySales.xsd">
  <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">
```

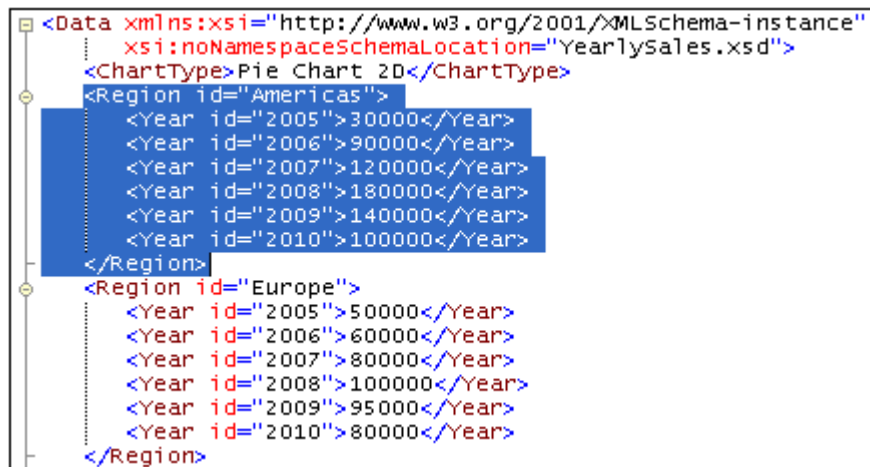
```

        <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 following cases are possible:

- If the cursor is placed anywhere inside the start tag (including in an attribute-value) or end tag of the `Data` element, or anywhere inside the `Data` element but not within a descendant node, the Source XPath will be: `/Data`
- If the cursor is placed anywhere inside the start tag (including in an attribute-value) or end tag of any `Region` element, or anywhere inside a `Region` element but not within a descendant node, the Source XPath will be: `/Data/Region`
- If the cursor is placed anywhere inside the start tag (including in an attribute-value) or end tag of a `Year` element, or anywhere inside a `Year` element, the Source XPath will be: `/Data/Region[ N ]/Year`. The predicate filter `[ N ]` selects the particular `Region` element inside which the selected `Year` element is. So the X-Axis and Y-Axis data selections will be restricted to `Year` elements of this particular `Region` element.
- If you wish to select just one `Region` element, say: `/Data/Region[ 1 ]`, then highlight this element, that is, the first `Region` element as shown in the screenshot below.



- Similarly, highlighting two `Region` elements will generate an XPath expression that selects these two `Region` elements only.

The Source XPath expression can be edited subsequently in the Select Columns dialog.

In Grid View, selection is done by clicking a node or marking a range. The Source XPath that is generated from the Grid View selection is as described above for Text View.

Include indices in XPath

The *Include Indices* check box determines whether predicate filters in the XPath expression are

used, whether these predicate filters are entered automatically at the time the Select Columns dialog is called or whether they are entered manually. For example, if the cursor is placed inside a descendant element of the first `Region` element of the document and the *Include Indices* check box is checked, then the automatically entered XPath expression will be, for example: `/Data/Region[1]/Year`. If the *Include Indices* check box were not checked, then the expression would be: `/Data/Region/Year`.

The *Include Indices* check box also determines whether any predicate entered manually is retained. Therefore, if you wish to use predicates in the Source XPath expression, you must check the *Include Indices* check box.

Implications of Source XPath selections

Note the following implications of the Source XPath selection.

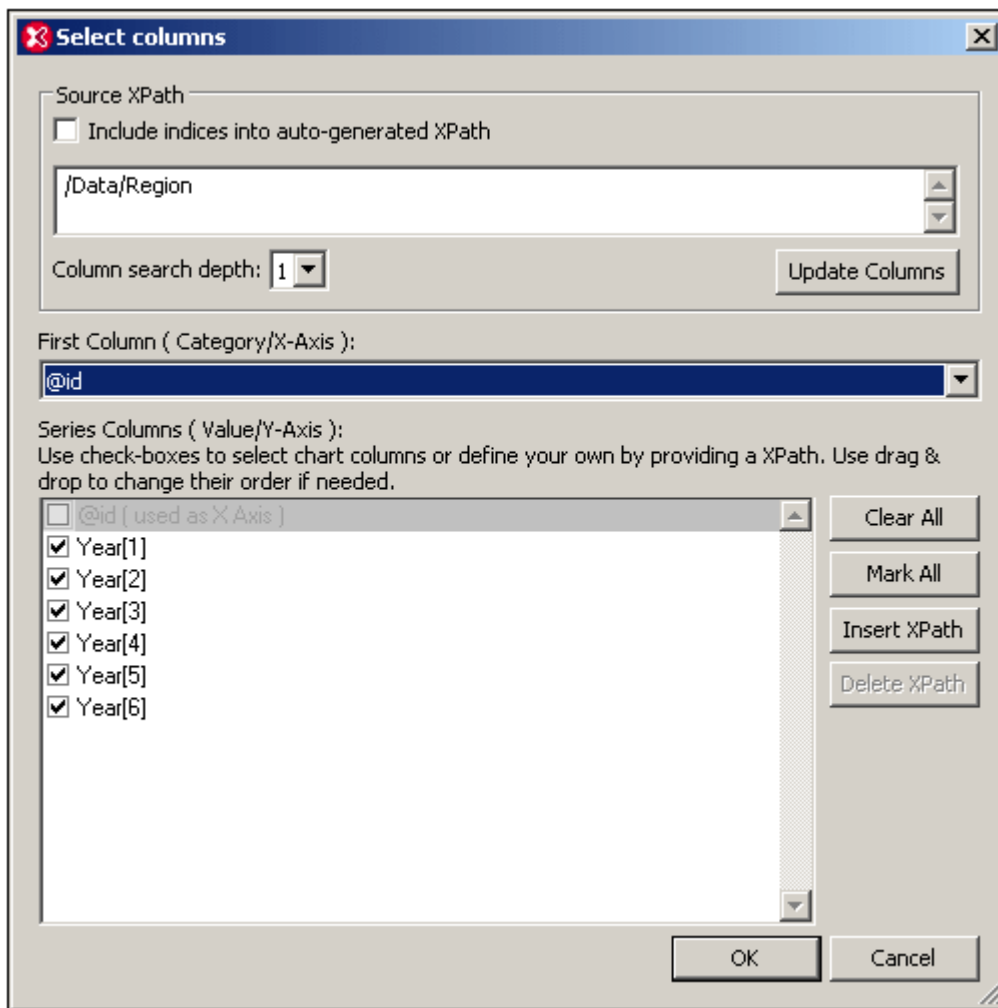
- The number of items in the sequence returned by the Source XPath determines the number of ticks on the X-Axis. The number of ticks on the X-Axis can be changed in only one other way besides by modifying the Source XPath: by selecting a number of labels for any series, which number is more than the number of ticks. See [X-Axis Selection](#) for more information about this scenario.
- The Source XPath node is the ancestor node for all nodes available for X-Axis and Y-Axis data selection and for all XPath expressions you may enter.
- As a result of the above two points, note that any change to the Source XPath expression affects not only the number of ticks on the X-Axis but also the context for any XPath expression related to the chart.

For example, here are the implications of some XPath expressions with respect to the XML document shown above.

- `/Data/Region`: Returns the three `Region` elements, so three ticks on the X-Axis. Each `Region` element will in turn be the context node for XPath expressions.
- `/Data/Region/Year`: Returns 18 `Year` elements, so 18 ticks on the X-Axis. Each `Year` element will in turn be the context node for XPath expressions.
- `/Data/Region[1]/Year`: Returns the six `Year` element children of the first `Region` element, so six ticks on the X-Axis. Each `Year` element of the first `Region` element will in turn be the context node for XPath expressions.
- `distinct-values(//Year/@id)`: Returns six items (the distinct values of the `Year/@id` attribute: 2005, 2006, 2007, 2008, 2009, 2010). However, since this XPath expression returns no node item, it cannot be a context node for any XPath expression. If the items in this sequence are to be used to target nodes in the XML document (using, say, the `current()` function (shorthand: `.`)), then the XPath expression using the current item must start from the document root in order for the context to be established. For example: `/Data/Region[1]/Year[@id eq .]`.

5.8.3 X-Axis Selection

The X-Axis selection is specified in the Select Columns dialog (*screenshot below*). This selection determines the labels that appear on the X-Axis. The labels can subsequently be edited in the Select Data dialog (*see below*).



Consider the following XML document example. (It is named `YearlySales.xml` and is available in the folder `C:\Documents and Settings\<username>\My Documents\Altova\XMLSpy2012\Examples\Tutorial.`) The cursor is placed in the start tag of the first `Region` element and the **New Chart** button of the Charts output window is clicked. The Select Columns dialog appears, with the Source XPath: `/Data/Region` (see screenshot above).

```
<?xml version="1.0" encoding="UTF-8"?>
<Data xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="YearlySales.xsd">
  <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>
  </Region>
</Data>
```

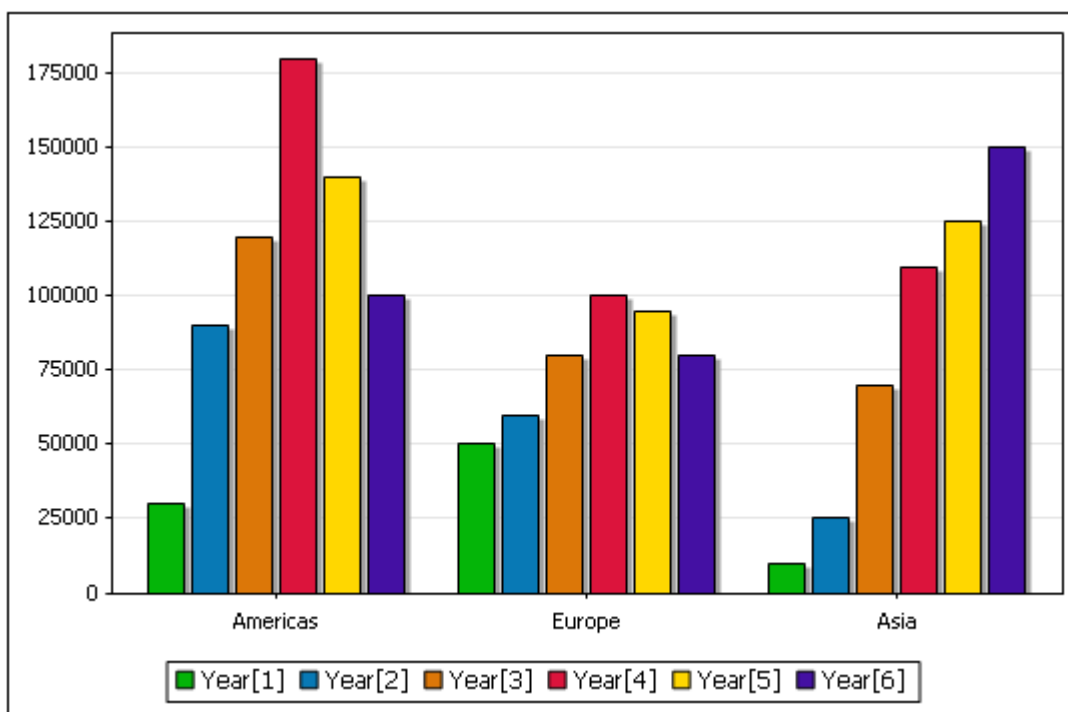
```

<Year id="2007">70000</Year>
<Year id="2008">110000</Year>
<Year id="2009">125000</Year>
<Year id="2010">150000</Year>
</Region>
</Data>

```

As explained in the [Source XPath](#) section, this Source XPath sets up a chart with three ticks on the X-Axis (because the Source XPath returns three items: the three `Region` elements). Since we want the labels of these three ticks in the chart to be the names of the three regions, we select the `@id` attribute in the combo box of the X-Axis selection (see *screenshot of Select Columns dialog above*).

To produce the chart data for each tick, each `Region` element is evaluated in turn. For each `Region` element, the `id` attribute generates the correct label for the X-Axis tick. The X-Axis would look something like in the screenshot below.

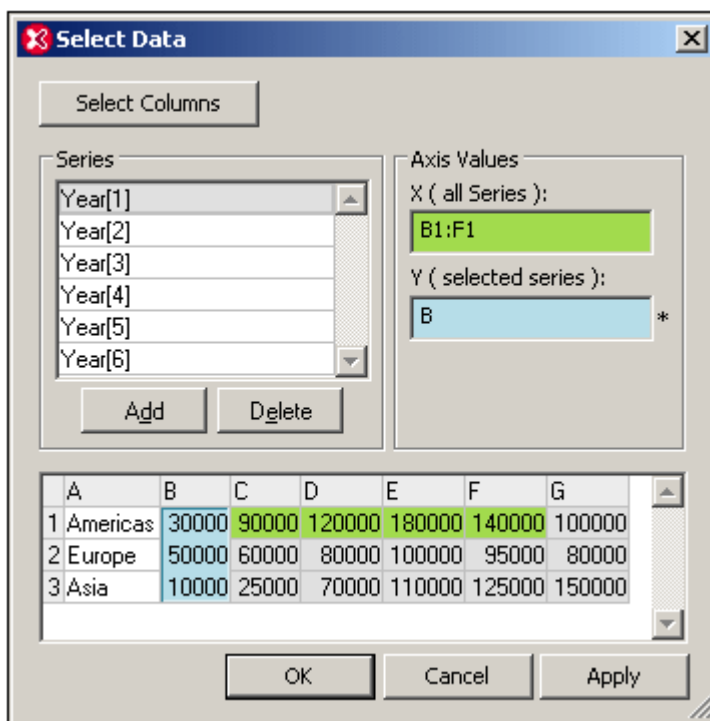


If another XPath expression is selected in the X-Axis combo box, then that expression is evaluated within the respective `Region` element context and the evaluated result will be the label of the respective tick. The *Auto-Enumerated* option generates a number sequence that corresponds to the tick number: the first tick will be numbered 1, the second 2, and so on.

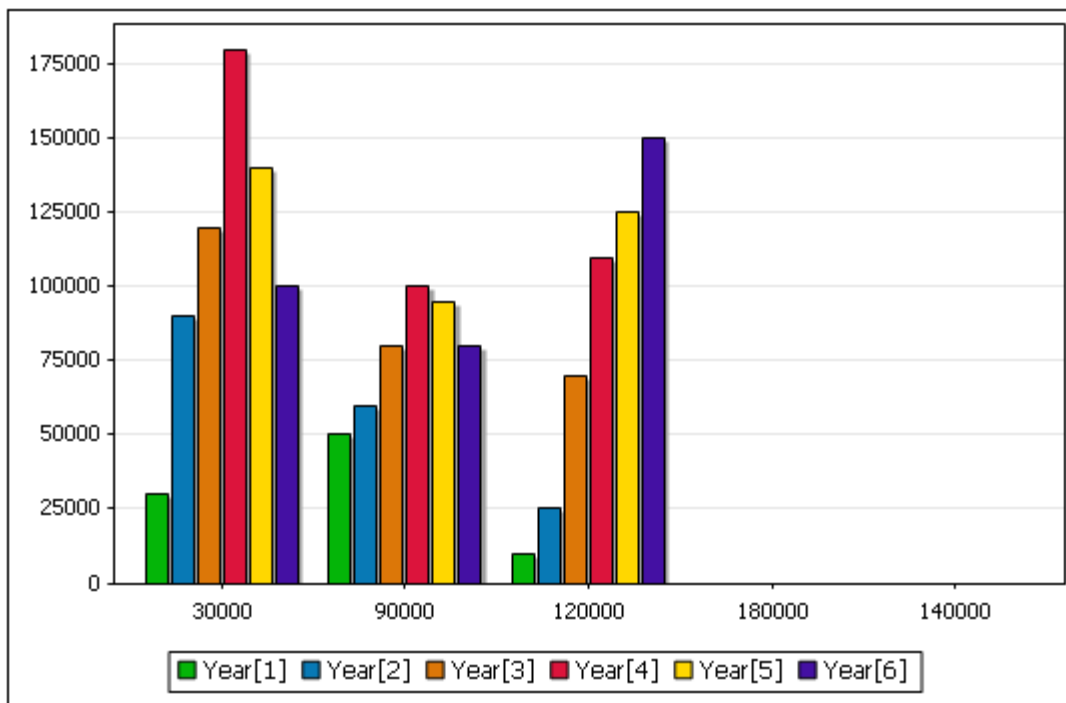
Modifying the X-Axis labels and the number of X-Axis ticks

The selection of labels for the X-Axis can be modified in the Select Data dialog (accessed by clicking the **Select Data** button of the Charts output window).

In the Select Data dialog shown in the screenshot below, for example, click in the X-Axis text box of the Axis Values pane. This enables the X-Axis selection to be modified. Now click the `B1` field and drag the mouse to `F1` to make the `B1: F1` selection. Click **OK** to see the new chart.



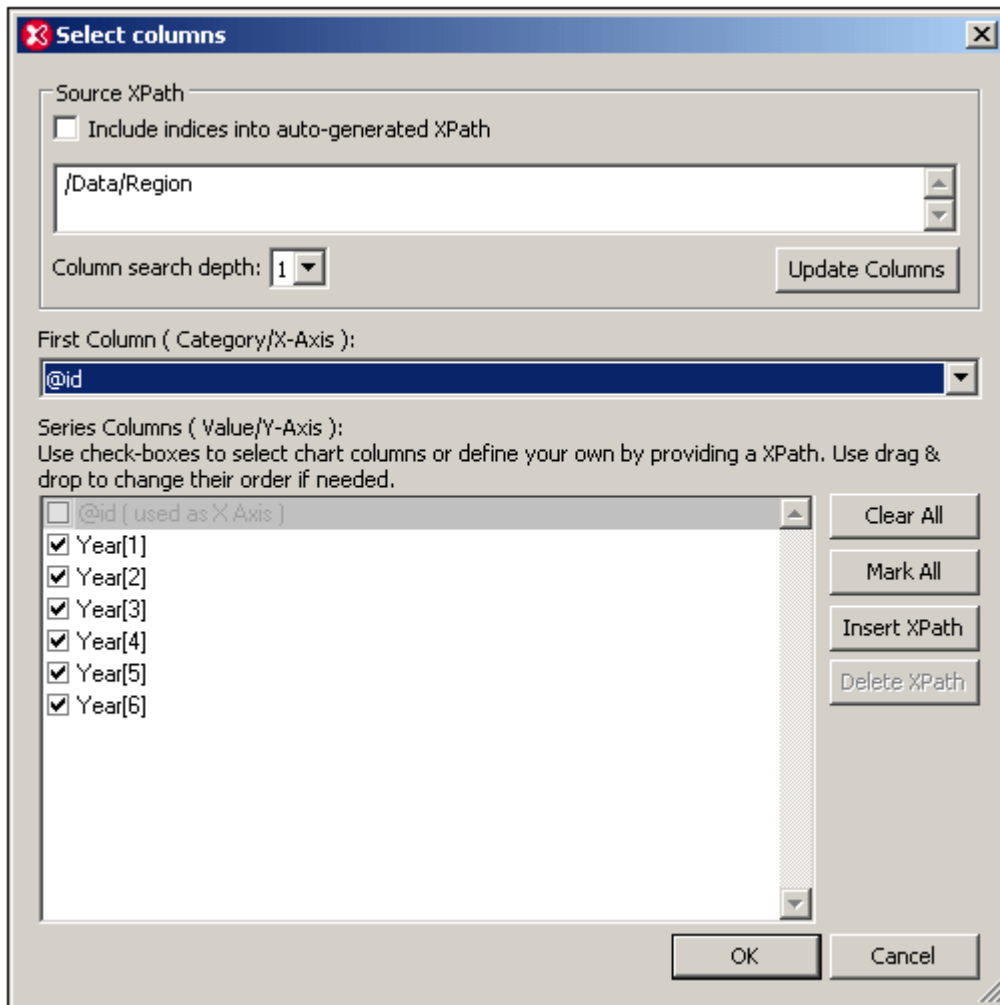
This selection will now provide the labels for the ticks, as shown in the screenshot below. Notice also that since the new selection contains five items, five ticks have been generated. However, only the first three are populated. This is because the Source XPath returns three nodes and these are the nodes that will be processed for the charts. These three nodes correspond to the [rows of the table](#) shown in the Select Data dialog. Note that the number of rows in the table can only be modified by changing the Source XPath.



For more information about how the Source XPath and X-Axis selections interact, see [How Chart Data Is Created](#).

5.8.4 Y-Axis Selection

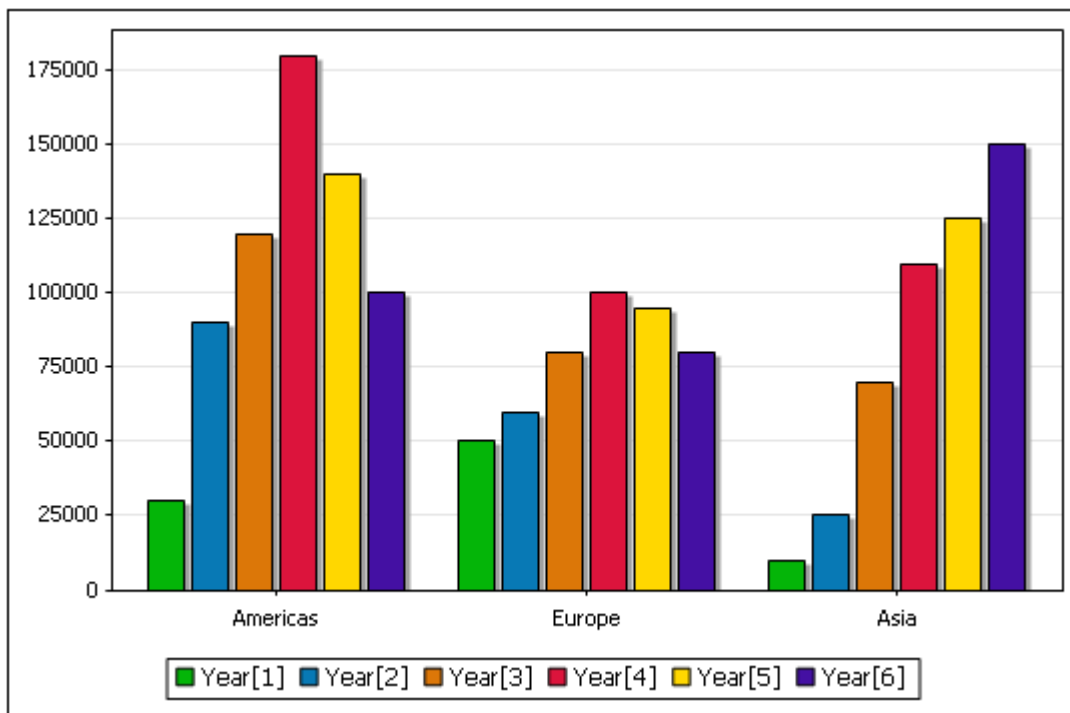
The Y-Axis (*see screenshot below*) is also known as the Series Axis.



The selection you make for this axis determines how many series are plotted for each X-Axis tick. If only one series is selected, then at each X-Axis tick, the value returned by the XPath expression for that series is plotted. If more series are selected, as in the screenshot below, in which six series are selected, then the chart data selection will be as in the table below. (The [context node](#) for the Y-Axis data selection is the respective `Region` element.)

Source XPath	X-Axis	Y-Axis (Series columns)					
Region[1]	@id	Year[1]	Year[2]	Year[3]	Year[4]	Year[5]	Year[6]
Region[2]	@id	Year[1]	Year[2]	Year[3]	Year[4]	Year[5]	Year[6]
Region[3]	@id	Year[1]	Year[2]	Year[3]	Year[4]	Year[5]	Year[6]

The resulting chart will look something like this:



There are three X-Axis ticks, labeled with the value of the respective `Region/@id` attributes. At each X-Axis tick, the XPath expression for each series is evaluated. In our example, for each X-Axis tick, each of the six series is evaluated and plotted. For example, the first series (`Year[ 1 ]`) is plotted for all three regions, so also `Year[ 2 ]` to `Year[ 6 ]`.

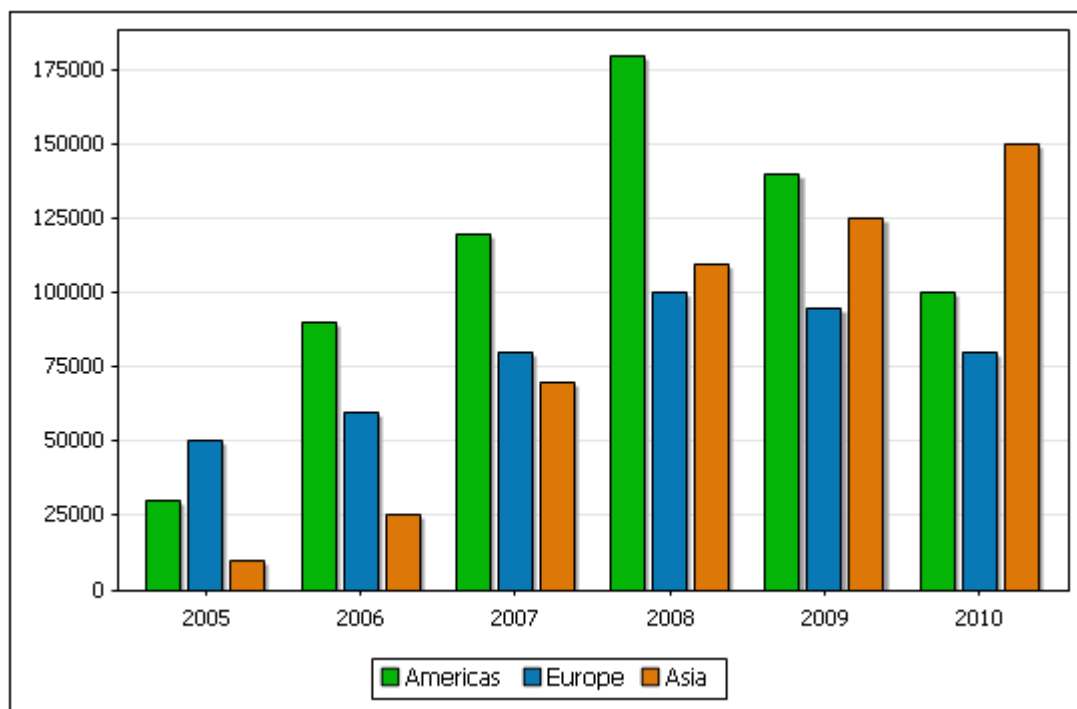
Note: Some charts, such as pie charts and single-bar charts, take only one axis. In a single-bar chart for example, each X-Axis tick will have just a single bar: that representing the single series. In a **pie chart**, the values of the single series will sum up to 100% of the pie, with each value being assigned to one X-Axis tick.

Y-Axis legends

The legends that appear below the chart are the names of the series. These names can be modified in the [Select Data dialog](#).

Switching the X-Axis and Y-Axis selections

In the example above, the regions are on the X-Axis and the yearly sales are plotted on the Y-Axis for each region; the `Year` elements are the series. But what if we wished to plot the years on the X-Axis and compare the regional sales for each year as in the bar chart below? We would need six X-Axis ticks (obtained via the Source XPath selection), then to label the X-Axis ticks with the respective years, and finally to select three series (for the regions), all of which will be represented at each X-Axis tick. The screenshot below, of the Select Columns dialog, shows how this data selection might be achieved.



Selecting the series

To make a node a series for the chart, check that node's check box. You can modify the Source XPath and the Column Search Depth to make the required node available in the Series pane. Alternatively, you can add an XPath expression to select a node, as in the screenshot below. See [Chart Example: Advanced](#) for a description of this scenario.

Reference

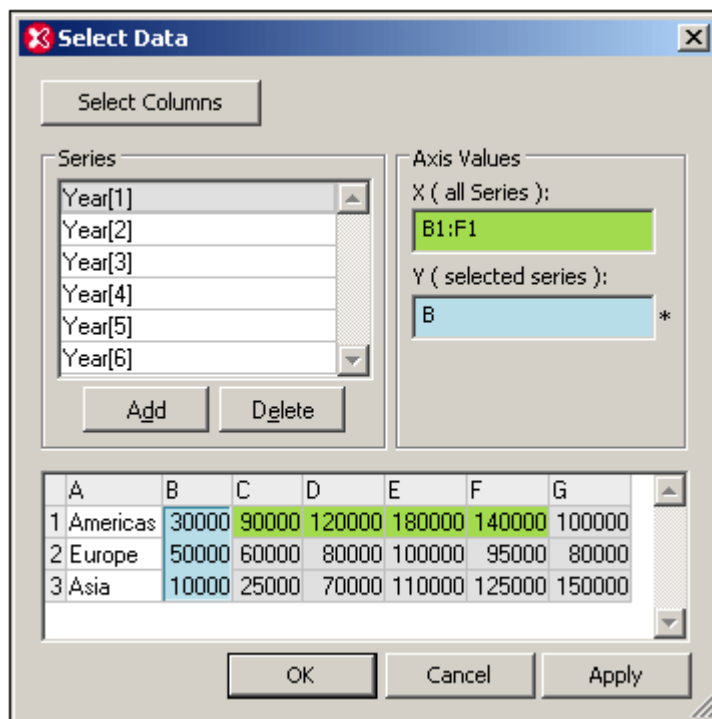
The XML document used for the example in this section is given here for reference. It is named `YearlySales.xml` and is available in the folder `C:\Documents and Settings\<username>\My Documents\Altova\XMLSpy2012\Examples\Tutorial`.

```
<?xml version="1.0" encoding="UTF-8"?>
<Data xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance "
  xsi:noNamespaceSchemaLocation="YearlySales.xsd">
  <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>
```

5.8.5 Chart Data

Clicking the **Select Data** button pops up the Select Data dialog (*screenshot below*), which consists of three panes: (i) the Series pane, (ii) the Axis Values pane, and (iii) the chart data table. Each of these is described below.

The **Select Columns** button pops up the [Select Columns dialog](#), in which you can change the Source XPath and modify the data selection for the X-Axis and Y-Axis.



Series pane

The series contained originally in the Series pane are those that were selected in the [Select Columns dialog](#). The series present in this pane when you click **OK** will be the series that appear in the chart. In the Series pane you can carry out three operations:

- *Add and delete series:* This enables you to control the number of series appearing in the chart.
- *Edit series names:* The names of series are the legends that appear in the chart.
- *Select data for individual series:* With a series selected in the Series pane, the X-Axis and Y-Axis data can be specified in the Axis Values pane. How to do this is described below.

Axis Values pane

The X-Axis and Y-Axis data can be specified in the respective text boxes in the Axis Values pane. When you click in either text box, the value in it can be edited; this is indicated by an asterisk to the right of the text box. The data selection is specified either as a range from the chart data table. A range can be either an entire column or row, or part of a column or a row. It can be entered via the keyboard (for example, A or 3 or B1: F1), or a range can be marked in the chart data table. To mark a range, select the first cell in the range and drag the cursor to the last cell in the range. To mark an entire column or row, select the column or row header, respectively.

The X-Axis selection determines the labels of the X-Axis nodes and applies to all series. It does not change the number of X-Axis ticks.

The Y-Axis selection determines which range of cells is to be used for the selected series. If the number of cells selected is less than the number of X-Axis ticks, then this series will be unrepresented for the latter X-Axis ticks. If the number of cells selected exceeds the number of X-Axis ticks, then additional ticks will be created. The extra ticks will be equal to the number of extra selected cells. The extra values will be represented for this series on the extra number of ticks.

Chart data table

The structure of the chart data table (at the bottom of the Select Data dialog) is obtained from the selections in the [Select Columns dialog](#).

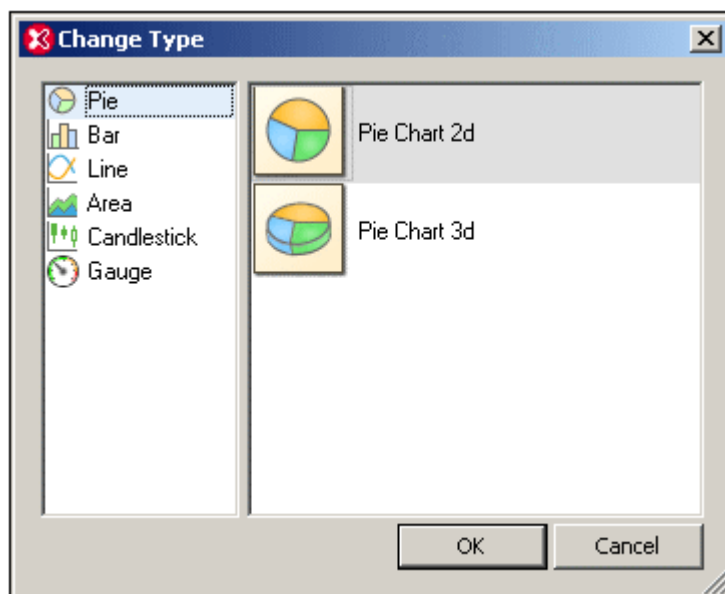
- The number of rows in the table is equal to the number of items in the sequence returned by the Source XPath.
- The columns are named starting from A. The purpose of this naming is to enable selection in the Axis Values pane (for example B1: F1).
- The first column is obtained by evaluating the X-Axis selection in the [Select Columns dialog](#) in the context of nodes returned by the Source XPath expression.
- All other columns except the first are obtained by evaluating the Y-Axis selections in the [Select Columns dialog](#). Each series in the Y-Axis selection of the [Select Columns dialog](#) corresponds to a column in the chart data table.

The chart data table can be viewed as a superset of data that is selected using the parameters in the [Select Columns dialog](#). From this superset, you can then select ranges of data you require (in the Axis Values pane) for individual series.

5.8.6 Chart Settings: Quick Reference

Chart type

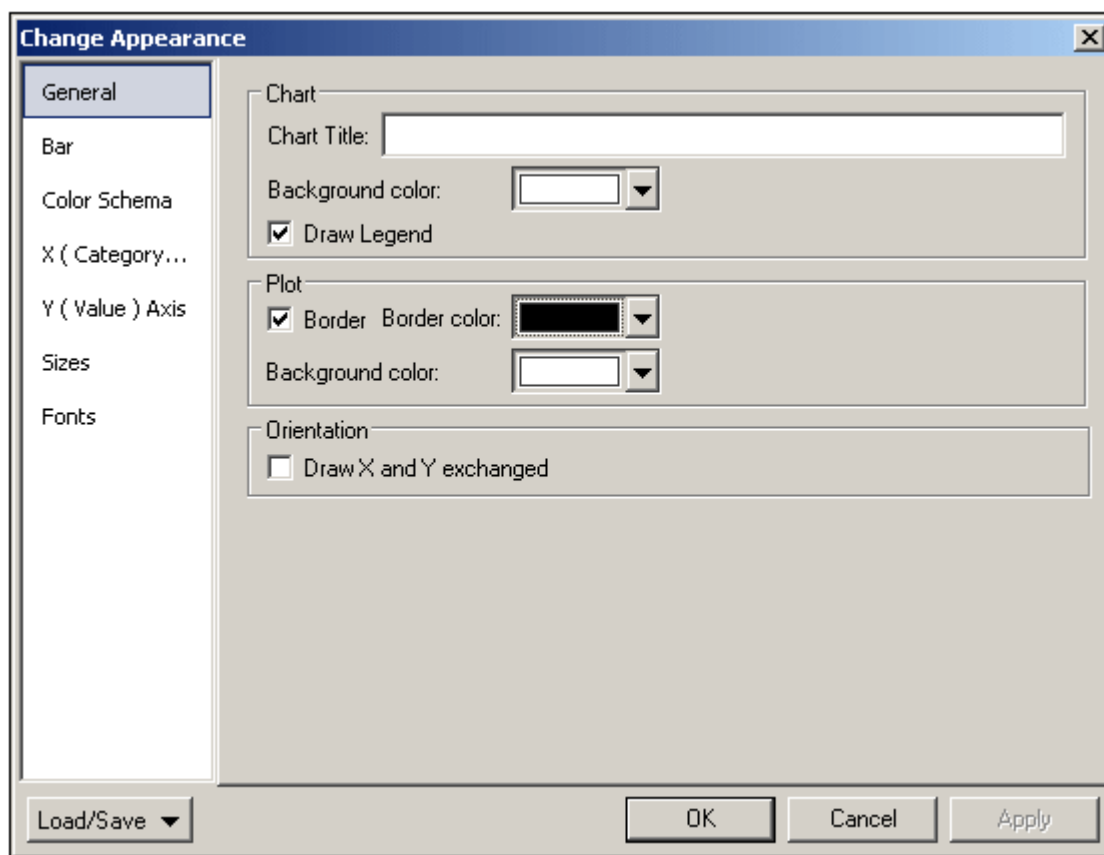
To select the chart type in the Change Type dialog (*screenshot below*), select the chart type you want and click **OK**. The Chart Type dialog is accessed by clicking the **Change Type** button.



After the chart type has been selected, chart settings (such as title, height, and width) must be made in the Change Appearance dialog (*screenshot below*) and the chart data must be specified. How data is selected for each chart type is described in the sections, [Creating a Chart](#) and [Chart Data](#).

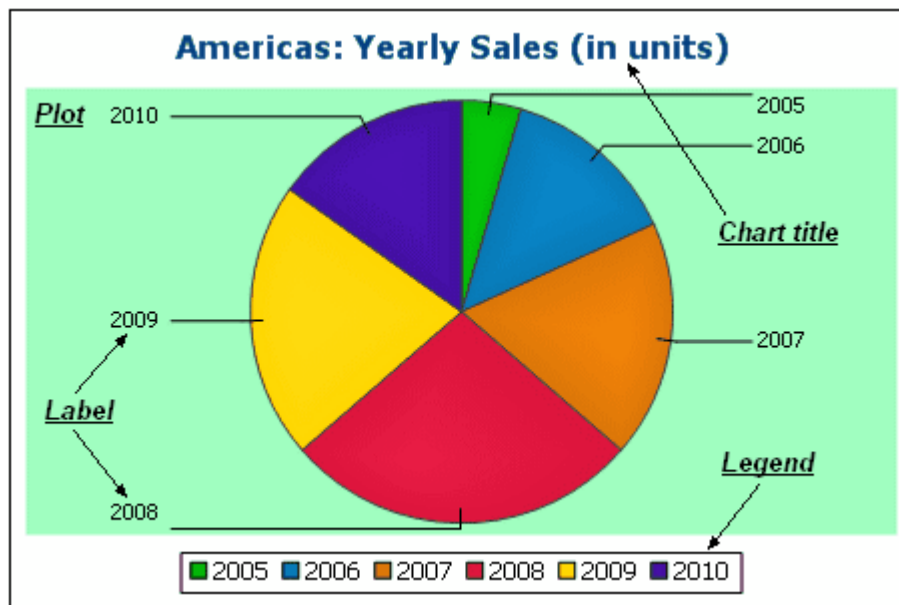
Chart appearance

Chart settings (such as the chart's title, color scheme, and font sizes) are made in the Change Appearance dialog (*screenshot below, which shows the Settings dialog of a bar chart*). This dialog is accessed with the Change Appearance button and the settings in it are different according to the chart type.



The various settings are organized into the following common tabs:

- **General:** The chart title (*see screenshot below*) can be edited in this tab, as well as the chart's background color and the plot's border and background color. In the screenshot below, the plot has been given a pale green background color. The legends are at the bottom of the chart; they explain the color codes in the chart and can be turned on or off in the dialog.



- **Color scheme:** Four predefined color schemes are available plus a user-defined color scheme. You can modify any of the color schemes by adding and/or deleting colors to a scheme. The color scheme selected in the Color Scheme tab will be used in the chart.
- **Sizes:** Sizes of various aspects of the chart can be set, either as pixels or as a percentage ratio.
- **Font:** The font properties of the chart title and of legends and labels can be specified in this tab. Sizes can be set as a percentage of the chart size or as pixels.

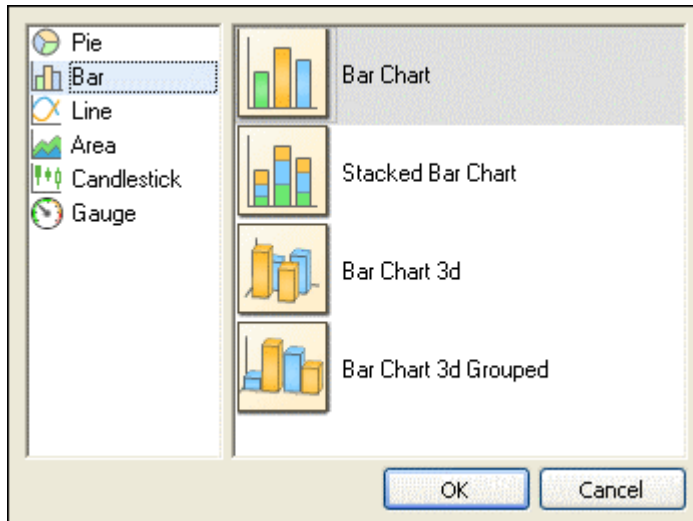
Additionally, each type of chart has settings specific to its type. These are listed below:

- **Pie charts:** Settings for: (i) the angle from which the first slice should be drawn; (ii) the direction in which slices should be drawn; (iii) the outline color; (iv) whether the colors receive highlights (in 3D pie charts: whether dropshadows and transparency are used); (v) whether labels should be drawn; and (vi) whether values and percentages should be added to labels and how many decimal places should be added to the percentages.
- **Bar charts:** Settings for: (*General*) Drawing the X and Y axes exchanged generates a horizontal bar chart (for 2D bar charts only); (*Bar*) Bar outlines and dropshadows (dropshadows in 2D bar charts only); (*X-Axis*) Label and color of the x-axis, and vertical gridlines; (*Y-Axis*) Label and color of the y-axis, horizontal gridlines, the range of values to be displayed, and the tick marks on the y-axis; (*Z-Axis, 3D only*) Label and color of the z-axis; (*3D*) the vertical tilt, horizontal rotation, and the width of the view.
- **Line graphs:** Settings for: (*General*) Drawing the X and Y axes exchanged; (*Line*) including the plot points or not; (*X-Axis*) Label and color of the x-axis, and vertical gridlines; (*Y-Axis*) Label and color of the y-axis, horizontal gridlines, the range of values to be displayed, and the tick marks on the y-axis.
- **Gauge:** Settings for: (i) the angle at which the gauge starts and the angular sweep of the scale (*Round Gauge only*); (ii) the range of the values displayed (*Round and Bar Gauges*); (iii) the interval and color of major and minor ticks (*Round and Bar Gauges*); (iv) colors of the dial, the needle, and the border.

5.8.7 Defining the Chart Type

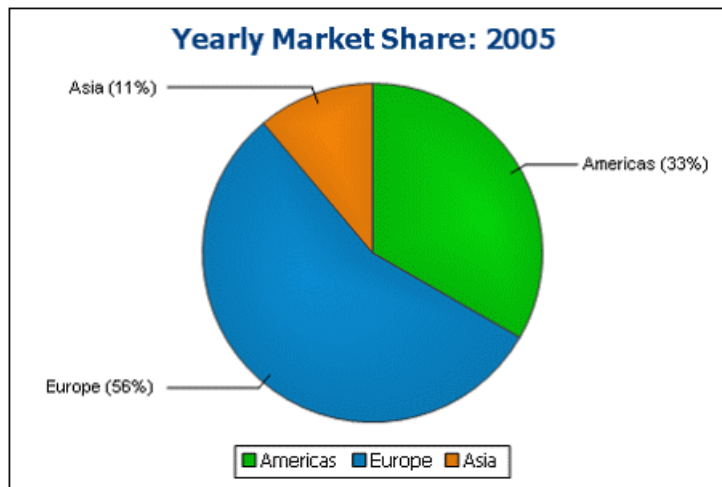
The type of a chart is defined in the **Change Type** dialog box which opens when you click the **Change Type** button in the toolbar of the Chart window. This resizable dialog box offers six different chart types (Pie, Bar, Value, Area, Candlestick and Gauge) as well as several sub-

types, such as 2d or 3d charts.



Pie Charts

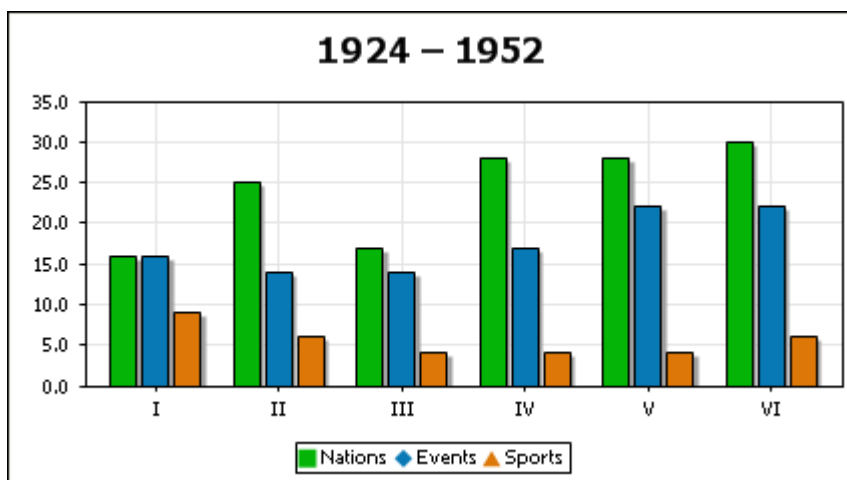
In Pie charts you can display the values of a single axis and use another axis to label the values.



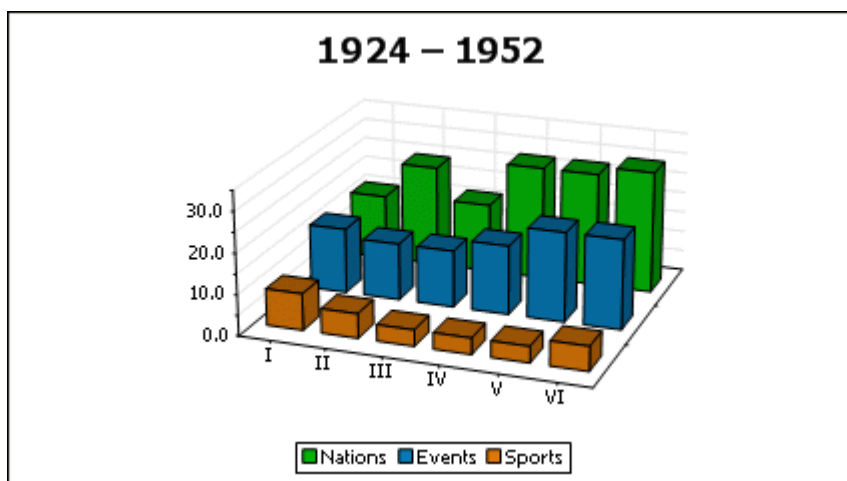
You can choose between 2d and 3d display and show a legend and/or labels (both legend and labels are on in the above screenshot).

Bar Charts

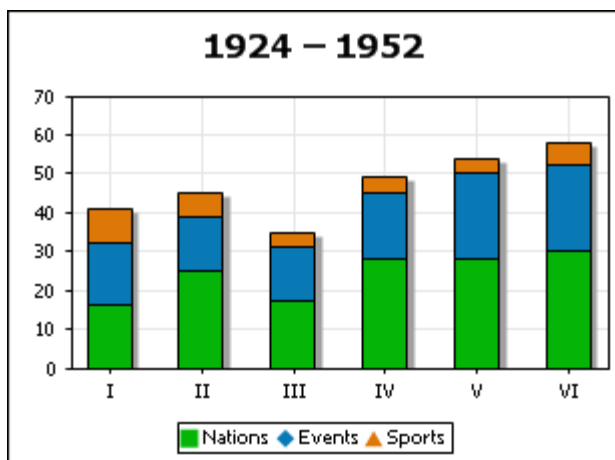
In Bar charts, you can use non-numerical values for the X-axis.



Bar charts can be displayed in 2d or 3d.



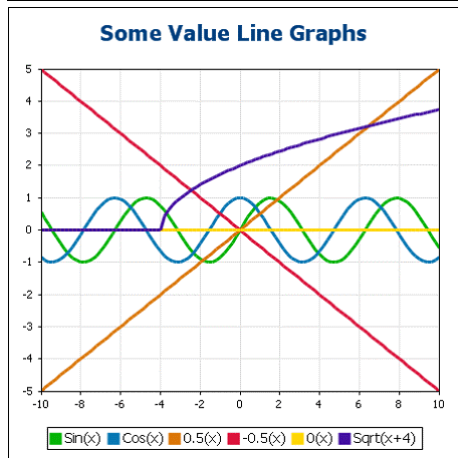
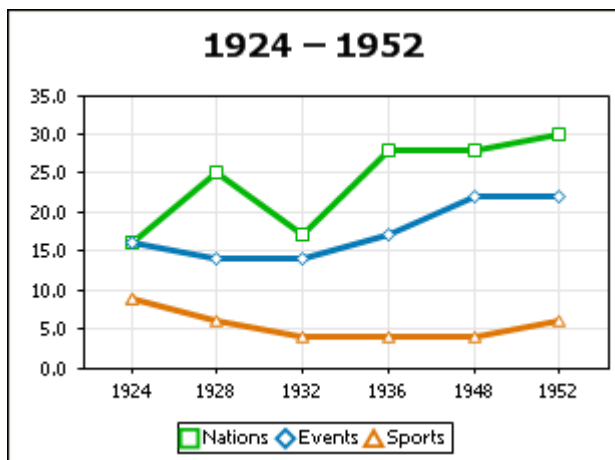
In addition, you can also choose to display stacked bar charts if you need to show the total of the series included in the chart.



Line Charts

The difference between a normal Line Chart (*screenshot below left*) and Value Line Chart (

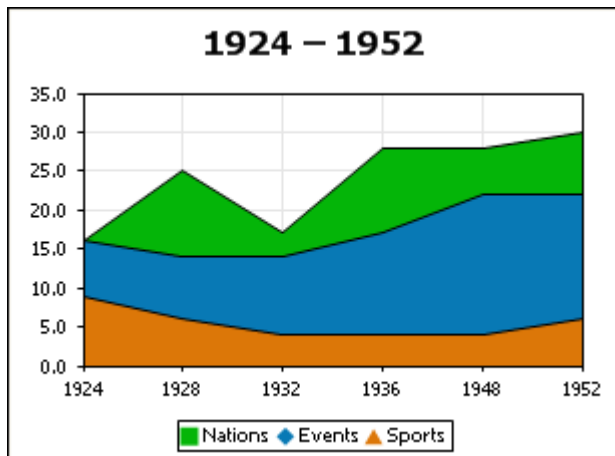
screenshot below right) is that in Value Line Charts you can use only numerical columns for the X-axis. If you need to display line charts with text values on the X-axis, choose "Line Chart" from the right pane of the **Change Type** dialog box.



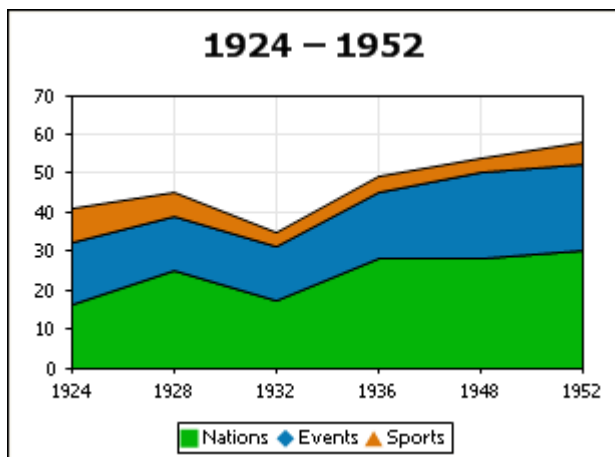
Note that in the screenshot above at left the X-axis now has the year as a unit and not the period as previously in the bar charts.

Area Charts

Area charts are a variation of line charts where the area below the line is also colored.

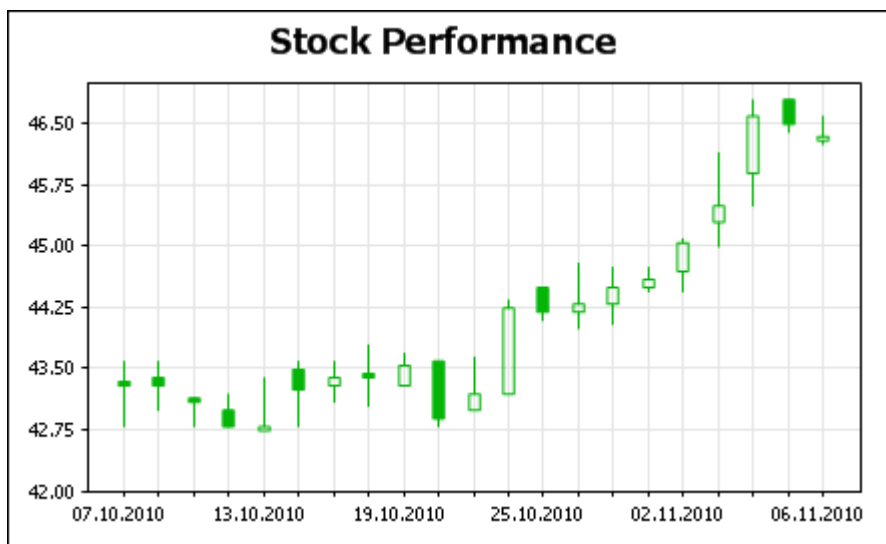


If you want to display the overall sum of several data series, you can choose the stacked version of area charts (*see screenshot below*).

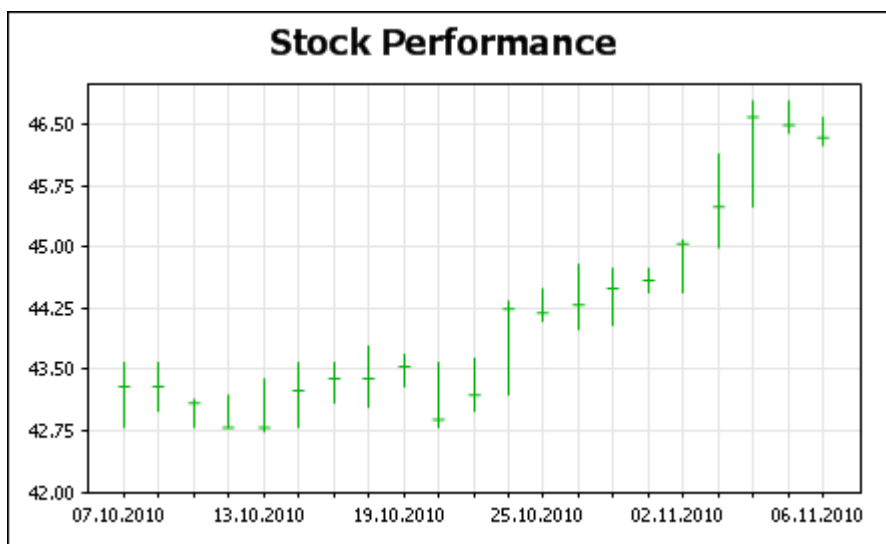


Candlestick Charts

A candlestick chart can be used to depict price movements of securities, derivatives, or currencies over a period of time. The chart provides an impression of how the asset developed, showing not only the closing price but also high and low and, optionally, the opening price per day. If you define four series (i.e., in this sequence: open, high, low, and close) the chart will look approximately like in the screenshot below.



The screenshot below shows a candlestick chart where only closing price, high and low have been defined.



Gauge Charts

Gauge charts are used to illustrate a single value and show its relation to a minimum and a maximum value. The screenshot below shows the number of participating nations in the Olympic Winter Games of 1976 in Austria in relation to the maximum of 80 participating nations in the Winter Games of 2006.



Changing the type of a chart

To change the type of a chart, click the **Change Type** button in the Charts Window. In the Change Type dialog that pops up, select the chart type you want and click **OK**.

5.8.8 Changing the Appearance of a Chart

You can change the appearance of a chart (e.g., title, legend, colors, fonts, etc.) in the **Change Appearance** dialog box, which can be called by clicking the corresponding button in the toolbar of the Chart window. Please note that this dialog box provides different options and features, depending on which chart type you have selected. For each type, [General](#), [Sizes](#), and [Fonts](#) sections are available and additional sections are provided for the individual types.

The **Change Appearance** dialog box provides a **Load/Save** button that, when clicked, displays a menu with the following options:

- **Set to default:** Rejects any changes that you have made to the settings and restores the default settings. Please note that this restores the default in all sections of the dialog box not only in the section that is currently displayed!
- **Load from file:** Shows the Windows **Open** dialog box where you can choose a file and import settings that have been previously saved.
- **Save to file:** Opens the Windows **Save As** dialog box where you can specify a file name and path where the settings should be stored. The settings are saved in an XML file which lists options that are different from the default settings.

General

The General section of the **Change Appearance** dialog box lets you define the title of the chart, add or remove a legend, and define background pictures and colors and—for bar, line, area, and candlestick charts—orientation of the chart.

The screenshot shows the 'Change Appearance' dialog box with the 'General' section selected. It is divided into three sub-sections: Chart, Plot, and Orientation. In the 'Chart' section, the 'Chart Title' is 'Nations participating in Olympic Wintergames'. The 'Background color' is set to 'Vertical Gradient' with two color swatches. The 'Image File' is 'D:\images\OlympicRings.jpg' and the 'Stretch' option is 'Stretched'. The 'Draw Legend' checkbox is checked. In the 'Plot' section, the 'Border' checkbox is checked with a red border color. The 'Background color' is set to 'Solid' with a white color swatch. The 'Image File' is 'D:\images\OlympicRings.jpg' and the 'Stretch' option is 'Stretched'. In the 'Orientation' section, the 'Draw X and Y exchanged' checkbox is unchecked.

Chart

Enter a descriptive title for your chart into the **Chart Title** field and select a background color for the entire chart from the drop-down list. You can choose a solid background, vertical gradient, or horizontal gradient and define start and end colors for the gradient, if applicable. In addition, or instead of a colored background, you can also define a background image and choose one of the available display options from the drop-down list:

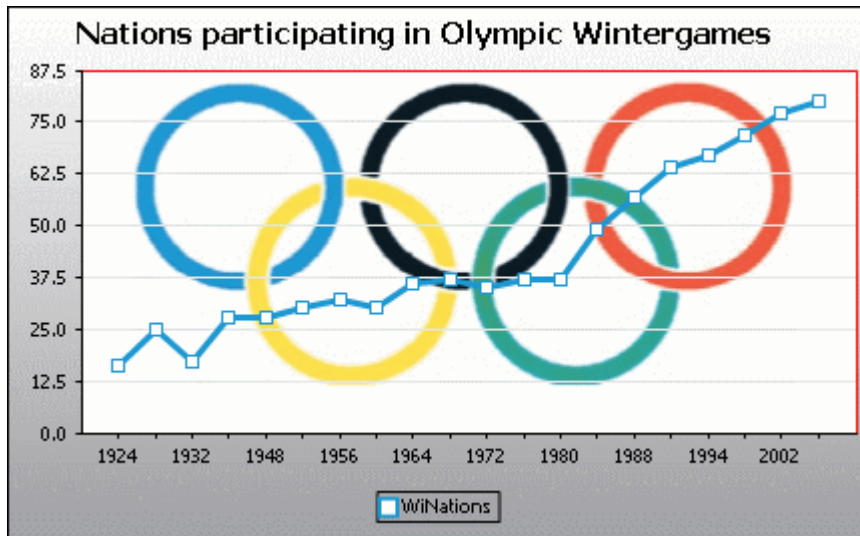
- **Stretched:** the image will be stretched to the height and width of the chart
- **Zoom to Fit:** the image will be fit into the frame of the chart and the aspect ratio of the image will be maintained
- **Center:** the image will be displayed in its original size in the center of the chart
- **Tiled:** if the image is smaller than the chart, duplicates of the image will be displayed to fill the background area

The **Draw Legend** check box is activated by default, clear the check box if you do not want to display a legend in your chart.

Plot

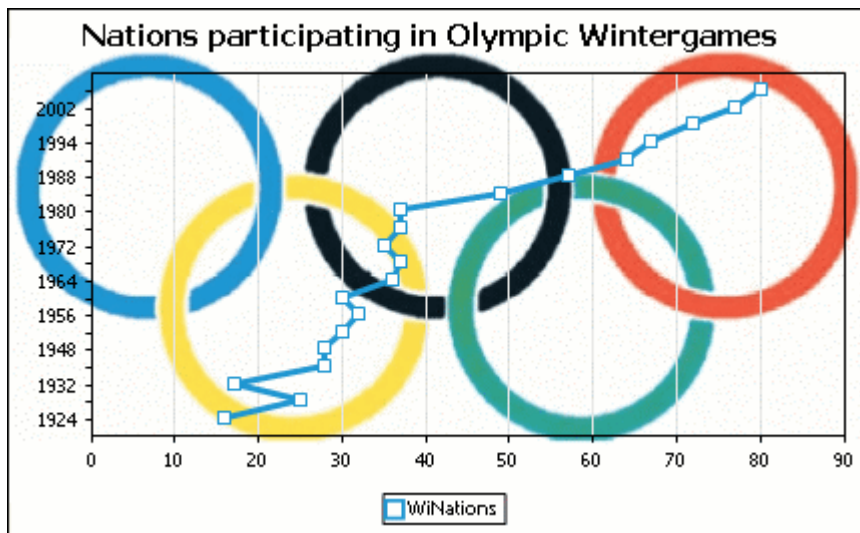
The Plot is the area where the actual data of the chart is displayed. You can draw a border around the plot and specify a different background color and/or image for the plot area. In the screenshot below, the background color of the chart has been changed to gray (vertical gradient) whereas the plot is still white, a red border has been drawn around it, and a

background image has been added.



Orientation

If you have a small series of large values it may be convenient to swap the X and Y axis for a better illustration. Note that in the screenshot below also the background color of the plot has been set to "Transparent" and the background image has been applied to the chart.



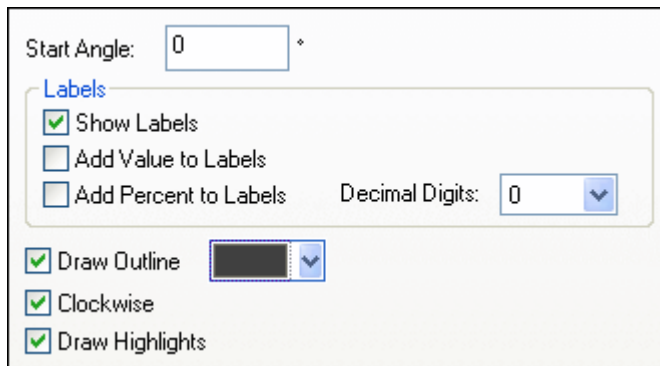
Note that this option is not available for pie and gauge charts.

Type-related Features

For each of the chart types, and even for the various sub-types, the **Change Appearance** dialog box provides a section where you can define the type-related features of the chart.

Pie chart

Most settings are the same for the 2d and 3d versions. In 2d pie charts, you can additionally draw highlights.

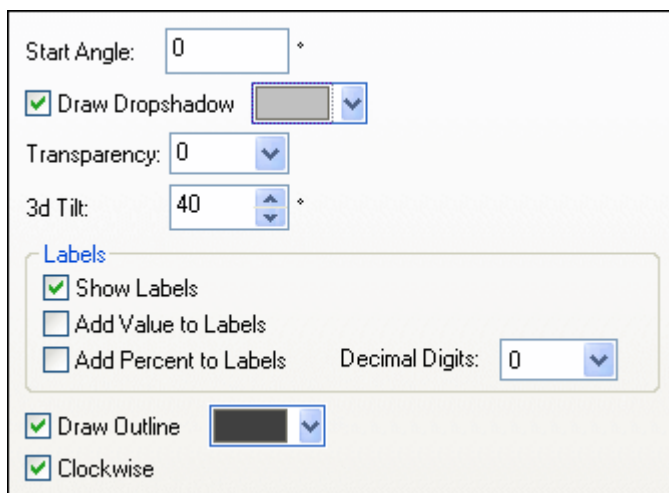


Start Angle: 0

Labels

- ☒ Show Labels
- ☐ Add Value to Labels
- ☐ Add Percent to Labels
- Decimal Digits: 0
- ☒ Draw Outline
- ☒ Clockwise
- ☒ Draw Highlights

In 3d pie charts, you can display drop shadows, add transparency and define the 3d tilt.



Start Angle: 0

- ☒ Draw Dropshadow
- Transparency: 0
- 3d Tilt: 40

Labels

- ☒ Show Labels
- ☐ Add Value to Labels
- ☐ Add Percent to Labels
- Decimal Digits: 0
- ☒ Draw Outline
- ☒ Clockwise

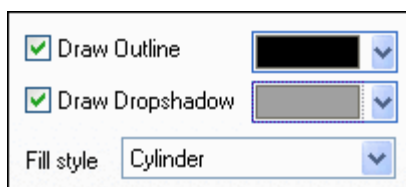
The `Start Angle` value defines where the first row of the selected column will be displayed in the chart. An angle of 0 degrees corresponds to 12 o'clock on a watch.

You can show labels in addition to, or instead of, the legend, add values and/or percentage to the labels, and define for the percentage values the number of decimal digits to be displayed.

The color that you can select next to the `Draw Outline` check box is used for the optional border drawn around the chart and the individual pie segments. The `Clockwise` check box allows you to specify whether the rows should be listed clockwise or counter-clockwise.

In 3d pie charts, you can draw a drop shadow and define its color, add transparency to the chart, and define the 3d tilt. In 2d charts, the `Draw Highlights` option adds additional structure to the chart.

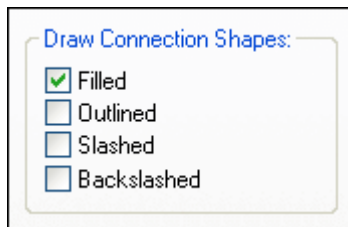
Bar chart



- ☒ Draw Outline
- ☒ Draw Dropshadow
- Fill style: Cylinder

For bar charts, you can add an outline to the bars and define its color. In 2d bar charts, you can also draw a drop shadow and define its color (this option is not available for 3d bar charts). By default, the shape of the bars resembles a cylinder, however you can also choose "Vertical Gradient" or "Solid" from the `Fill style` drop-down list.

Line chart

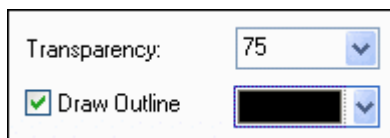


To draw connection shapes that mark the values in line charts, you need to activate at least one check box in the Draw Connection Shapes group box. You can use five different shapes to mark a series: square, rhomb, triangle, inverted triangle, and circle. If there are more than five series in your chart you can combine the connection shapes by selecting more than one option in the Draw Connection Shapes group box. In the screenshot below, both **Filled** and **Slashed** have been selected and the **Slashed** type is used for the sixth series and beyond.

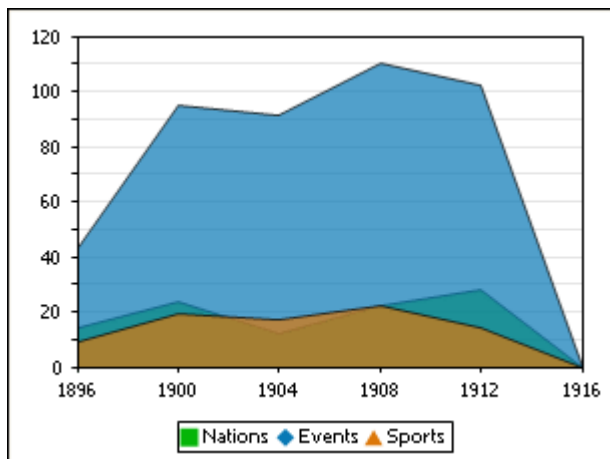


Connection shapes are available for both line charts and value line charts.

Area chart



Among the properties that you can change for area charts is transparency; this way you can prevent that one series is hidden by another series in the chart. In addition, you can add an outline to the individual data areas and define its color (see *screenshot below*).



Candlestick chart

Fill Color when Close > Open

☒ Unfilled

☐ [Color Picker]

Fill Color when Open > Close

☒ Use Series color

☐ [Color Picker]

If both opening and closing value are defined as series, you can choose the colors and whether or not the candle should be filled if the closing value is greater than the opening value.

Gauge chart

Angles

Start: 225 Sweep: 270

Value Range

Start: 0 End: 100

Major Ticks

Interval: 10 Color: [Green]

Minor Ticks

Interval: 5 Color: [Green]

Colors

Dial fill: [Black] Border: [Gray]

Needle: [Red] Needle Base: [Black]

The `Start` value defines the position of the 0 mark and the `Sweep` value is the angle that is used for display. In the Value Range group box you can define the minimum and maximum values to be displayed. Tick marks are displayed with (major ticks) or without (minor ticks) the corresponding value; you can define separate colors for them. In the Colors group box you can define colors for the dial fill, needle, needle base (hides the first part of the needle in the center of the chart), and the border that surrounds the chart.

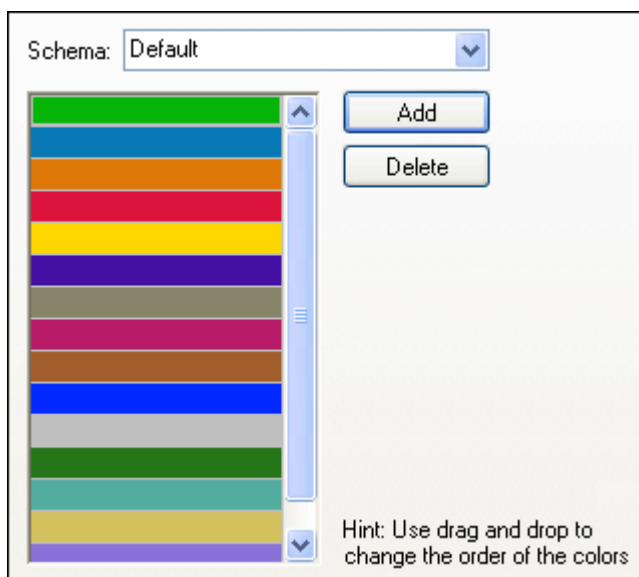
Colors

Depending on the chart type you have selected, XMLSpy provides two different sections for the definition of colors to be used in charts:

- Color Schema for pie, bar, line, area, and candlestick charts
- Color Range for gauge charts

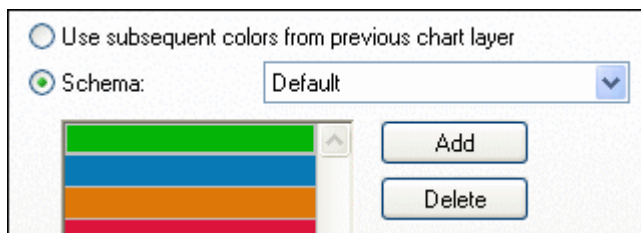
Color Schema

The Color Schema section of the **Change Appearance** dialog box provides four predefined color schemas (i.e., default, grayscale, colorful, and pastel) that can be customized; in addition you can also define your own color schema from scratch.



The top color will be used for the first series, then the second color and so on. You can change the order or the colors by selecting a color and dragging it to its new positions with the mouse. To add a new or delete an unwanted color, click the corresponding button. In candlestick charts, only the first color will be used.

If you have appended one or several layers of overlay charts to a Charts window, the Color Schema section of the **Change Appearance** dialog box contains the additional radio button *Use subsequent colors from previous chart layer* which is activated by default.



When the radio button is activated, the color schema from the previous layer will be used and you cannot choose a separate color schema for the overlay. The series of the active layer will be displayed using subsequent colors from the color schema of the previous layer. This way, all series of the Charts window have different colors and can therefore be distinguished more easily.

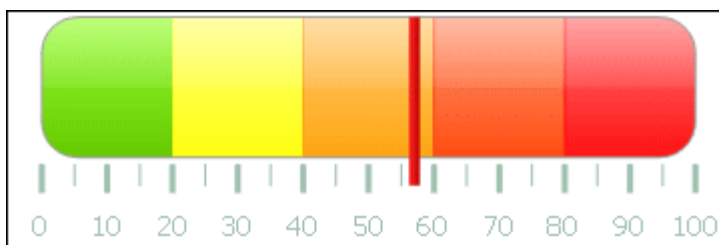
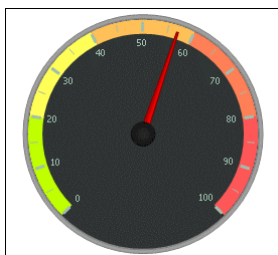
You can break this link on any additional layer that you add and choose a different color schema that then can also be re-used in subsequent layers.

Color Range

In gauge charts, you can customize the appearance of the gauge by applying colors to certain value ranges.

Starting from	Fill with color	Color
0	☒	
20	☒	
40	☒	
60	☒	
80	☒	

The definition shown in the screenshot above will appear in the gauge charts as follows:



X-Axis

In the X-axis section of the **Change Appearance** dialog box, you can enter a label for the axis, and define colors for the axis and the grid lines (if displayed). You can also define whether or not you want to display tick marks and axis values. This section is the same for all bar, line, area, and candlestick charts.

Label

Line

Gridlines
☐ Show Gridlines

Tick Drawing
☒ Show Ticks
☒ Show Values

Axis Position
 At Value / On Category Number:

In Value Line Charts however, you can also define the value range, and define at what interval tick marks should be displayed.

Label

The text entered into the `Label` field will be printed below the axis as a description of the X-axis.

Range

By default, the `Auto` radio button is selected in the Range group box. If you want to display a fragment of the chart in greater detail, activate the `Manual` radio button and enter minimum and maximum values into the respective fields.

If the column that is used for the X-axis does not include zero, you can deactivate the `Include zero` check box and the X-axis will start with the minimum value that is available in the series.

Line

The axis is displayed in the color that you choose from the `Line` drop-down-list. You can use one of the preselected colors, or click the **Other color...** button to choose a standard color or define a custom color. Click the **Select...** button on the Custom tab and use the pipette to pick a color that is displayed somewhere on your screen.

Grid lines

If the `Show Grid lines` check box is activated, you can choose a color from the corresponding drop-down list box.

Tick Interval

If you are not satisfied with the default tick marks, you can activate the `Manual` radio button in the Tick Interval group box and enter the difference between the individual tick marks into the corresponding field.

Tick Drawing

You can switch the display of tick marks on the axis and/or axis values on or off.

Axis Position

From the drop-down list, you can choose the position where the axis is to be displayed. When selecting "At Value / On Category Number", you can also position the axis anywhere within the plot.

Y-Axis

In the Y-axis section of the **Change Appearance** dialog box, you can enter a label for the axis, define colors for the axis and the grid lines (if displayed), define the value range, and decide if and where tick marks should be displayed and whether or not you want to show the axis values. This section is the same for all bar and line charts.

The screenshot shows the Y-Axis configuration section of a dialog box. It contains several sub-sections with labels in blue text:

- Label:** A text input field.
- Range:** Includes radio buttons for **Auto** (selected) and **Manual**. Next to **Auto** is a checked **Include Zero** checkbox. Next to **Manual** are input fields for **Min:** and **Max:**.
- Line:** A color selection dropdown menu showing a black color swatch.
- Gridlines:** Includes a checked **Show Gridlines** checkbox and a color selection dropdown menu.
- Tick Interval:** Includes radio buttons for **Auto** (selected) and **Manual**, with an input field next to **Manual**.
- Tick Drawing:** Includes checked checkboxes for **Show Ticks** and **Show Values**.
- Axis Position:** Includes a dropdown menu set to **Left/Bottom** and a text input field for **At Value / On Category Number:** with the value **0**.

Label

The text entered into the `Label` field will be printed to the left of the axis as a description of the Y-axis.

Range

By default, the `Auto` radio button is selected in the Range group box. If you want to display a fragment of the chart in greater detail, activate the `Manual` radio button and enter minimum and maximum values into the respective fields.

If the column that is used for the X-axis does not include zero, you can deactivate the `Include zero` check box and the X-axis will start with the minimum value that is available in the series.

Line

The axis is displayed in the color that you choose from the `Line` drop-down-list. You can use one of the preselected colors, or click the **Other color...** button to choose a standard color or define a custom color. Click the **Select...** button on the Custom tab and use the pipette to pick a color that is displayed somewhere on your screen.

Grid lines

If the `Show Grid lines` check box is activated, you can choose a color from the corresponding drop-down list box.

Tick Interval

If you are not satisfied with the default tick marks, you can activate the `Manual` radio button in the Tick Interval group box and enter the difference between the individual tick marks into the corresponding field.

Tick Drawing

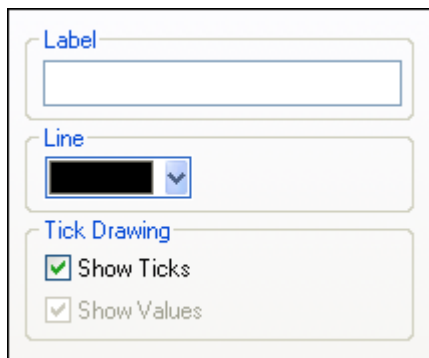
You can switch the display of tick marks on the axis and/or axis values on or off.

Axis Position

From the drop-down list, you can choose the position where the axis is to be displayed. When selecting "At Value / On Category Number", you can also position the axis anywhere within the plot.

Z-Axis

In the Z-axis section of the **Change Appearance** dialog box, you can enter a label for the axis, define colors for the axis, and decide whether or not you want to show tick marks on the axis. This section is the same for all 3d bar charts (Bar Chart 3d and Bar Chart 3d Grouped).

The image shows a screenshot of the Z-Axis configuration section from a software dialog box. It contains three main groups: 'Label' with a text input field; 'Line' with a color selection dropdown menu currently showing black; and 'Tick Drawing' with two checked checkboxes labeled 'Show Ticks' and 'Show Values'.

Label

The text entered into the `Label` field will be printed to the right of the axis as a description of the Z-axis.

Line

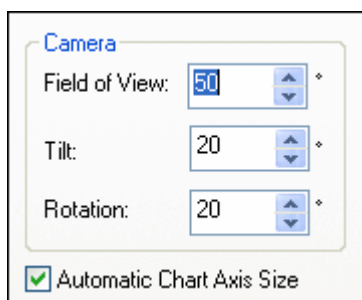
The axis is displayed in the color that you choose from the `Line` drop-down-list. You can use one of the preselected colors, or click the **Other color...** button to choose a standard color or define a custom color. Click the **Select...** button on the Custom tab and use the pipette to pick a color that is displayed somewhere on your screen.

Tick Drawing

You can switch the display of tick marks on the axis on or off.

3d Angles

In 3d bar charts you can customize the 3d appearance of the chart in the 3d Angles section of the **Change Appearance** dialog box.



The `Tilt` value determines the rotation around the X-axis, whereas the `Rotation` value defines the rotation around the Y-axis. You can automatically adapt the size of the chart axis to the Chart window width by activating the corresponding check box.

If the `Automatic Chart Axis Size` check box is activated, XMLSpy will automatically calculate the optimum size of the X-axis as well as the Y-axis for the current Chart window size. The width and height of the chart will change dynamically when you resize the Chart window.

Sizes

In the **Sizes** section of the **Change Appearance** dialog box, you can define different margins as well as the size of axis and gauge ticks. Note that not all the properties listed below are available for all chart types.

General

Outside margin	Space between the plot and the edge of the Chart window.
Title to Plot	Space between the chart title and the upper edge of the plot.
Legend to Plot	Space between the lower edge of the plot and the legend.

Pie

Plot to Label	In pie charts, the space between the most left and right edge of the pie and its labels.
Pie Height	In 3d pie charts, the height of the pie.
Pie Drop Shadow	In 3d pie charts, the length of the shadow (if it is activated in the Pie section).

X-Axis

X-Axis to Axis Label	In bar and line charts, the space between the X-axis and its label.
X-Axis to Plot	In 2d bar charts and line charts, the space between the X-axis and the plot.
X-Axis Tick Size	In bar and line charts, the length of the ticks on the X-axis.

Y-Axis

Y-Axis to Axis Label	In bar and line charts, the space between the Y-axis and its label.
Y-Axis to Plot	In 2d bar and line charts, the space between the Y-axis and the plot.
Y-Axis Tick Size	In bar and line charts, the length of the ticks on the Y-axis.

Z-Axis

Z-Axis to Axis Label	In 3d bar charts, the space between the Z-axis and its label.
Z-Axis Tick Size	In 3d bar charts, the length of the ticks on the Z-axis.

Line Drawing

Connection Shape	In line charts, the size of the squares that mark the values in the chart.

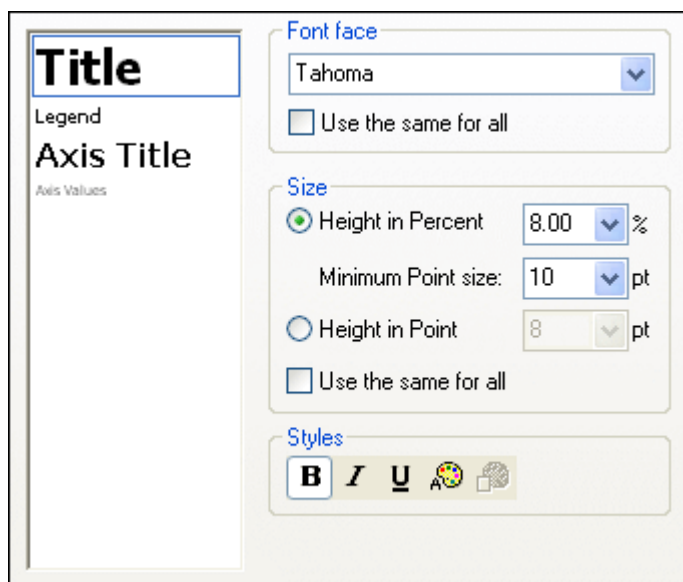
3d Axis Sizes

Manual X-Axis Size of Base	In 3d bar charts, defines the relation between the length of the X-axis and the Chart window size. Please note that the <code>Automatic Chart Axis Size</code> check box in the 3d Angles section must be deactivated, otherwise the size will still be calculated automatically.

Manual Y-Axis Size of Base	In 3d bar charts, defines the relation between the length of the Y-axis and the Chart window size. Please note that the <code>Automatic Chart Axis Size</code> check box in the 3d section must be deactivated, otherwise the size will still be calculated automatically.
Z-Axis Series Margin	In 3d bar charts, the distance on the Z-axis between the individual series.
Gauge	
Border Width	In round gauge charts, the width of the border around the gauge.
Gauge Ticks	
Border to Tick Distance	In round gauge charts, the space between the inner edge of the border and the ticks that mark the values.
Major Tick Length	In round gauge charts, the length of the major ticks (i.e., ticks that show a label).
Major Tick Width	In round gauge charts, the width of the major ticks (i.e., ticks that show a label).
Minor Tick Length	In round gauge charts, the length of ticks that do not have a value displayed.
Minor Tick Width	In round gauge charts, the width of ticks that do not have a value displayed.
Gauge Needle	
Needle Length	In round gauge charts, the length of the needle. (Note the the percentage is calculated from the diameter of the gauge, so if you choose a value greater that 50%, the needle will point to somewhere outside the gauge!)
Needle Width at Base	In round gauge charts, the width of the needle at the center of the gauge.
Needle Base Radius	In round gauge charts, the radius of the base that covers the center of the gauge.
Gauge Color Range	
Border to Color Range Distance	In round gauge charts, the space between the inner edge of the border and the outer edge of the color range .
Color Range Width	In round gauge charts, the width of the customizable color range. (Note that the percentage is calculated from the diameter of the gauge!)

Fonts

The Fonts section of the **Change Appearance** dialog box lets you configure fonts for objects in the Chart window.



Font settings

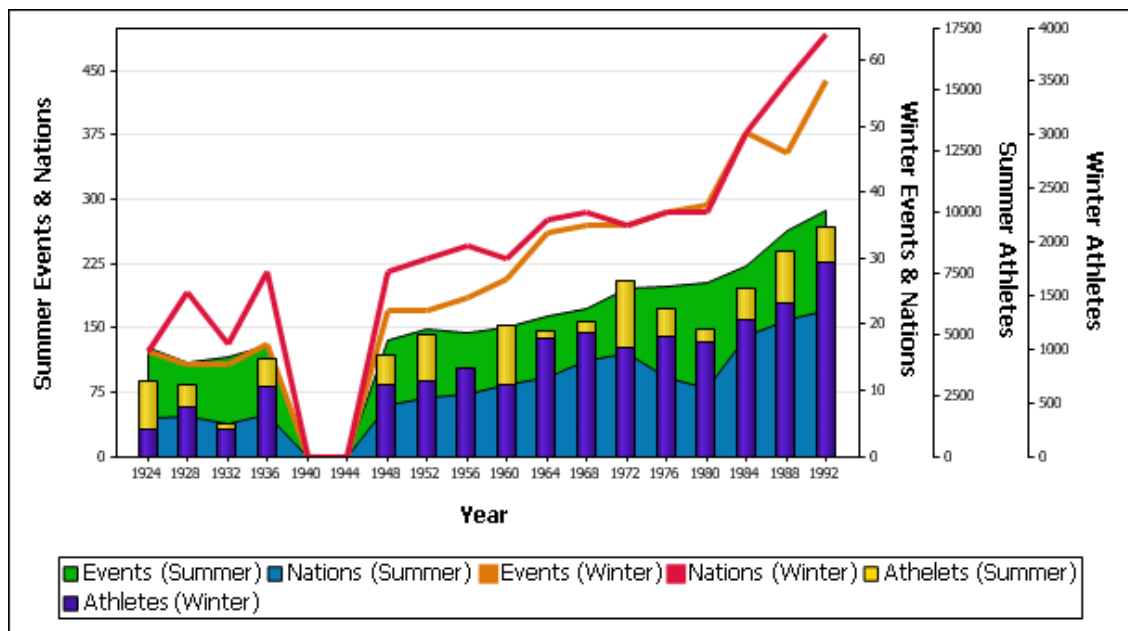
You can choose the font face, size, and style for the individual elements displayed in the Chart window. You can define the size as a percentage of the chart size and define a minimum size in points, or specify an absolute value (in points). To apply the same font and/or size to all text elements, activate the respective *Use the same for all* check box.

The element names in the list box are defined as follows:

- **Title:** The name of a chart
- **Legend:** The key to the colors used in the chart
- **Labels:** The designation of the pies of a pie chart
- **Axis Title:** The name of the X, Y, and Z axis in a bar or line chart
- **Axis Values:** The units displayed on an axis in a bar or line chart
- **Tick Values:** The units displayed on a gauge chart

5.8.9 Chart Overlays

For some chart types (Bar Chart, Stacked Bar Chart, Line Chart, Area Chart, Stacked Area Chart, and Candlestick Chart), you can display several charts as overlays in one Charts window. This way you can combine charts of different type containing data even from different in one Charts window.



A separate Y axis will be displayed for each overlay chart and can be labeled. Note that an overlay will always be displayed in front of the existing chart(s), you should, therefore be careful if you combine different chart types in one Charts window as, for example, area charts displayed as overlay may hide bar or line charts. By default, subsequent colors from the color schema of the Main chart will be used for the overlay. You can, however, also deactivate the *Use subsequent colors from previous chart layer* radio button in any layer and define a different color schema for this particular layer. Note that this color schema will be used for subsequent layers if these have the *Use subsequent colors from previous chart layer* radio button activated.

Chart overlays are managed by clicking the **Overlays** button in the menu bar of the Charts window; this will open the **Overlays** menu where you can add and remove overlays and select the active chart. Note that any changes you make to chart type and/or appearance, and in the

Select data dialog box apply only to the active chart!

To create a chart overlay:

1. Create a chart (the Main Chart).
2. In the Charts output window, click the **Overlays** button and choose **Append** from the drop-down menu.
3. Do one of the following:
 - Create a new chart from the existing Text View or Grid View, respectively.
 - Open a new XML document, select the node or nodes that you wish to use as the context node for the data selection, right-click and choose **New Chart...** from the context menu.

The new chart is displayed as overlay of the existing chart(s) in the same Charts output window.

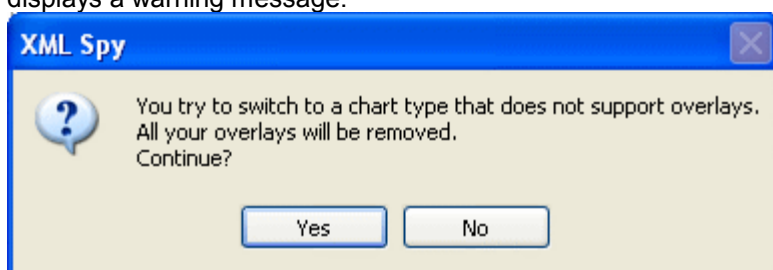
4. In the **Change Type** dialog box, which appears automatically for the first overlay of each tab in the Charts output window, define the type of the overlay chart.
5. Optionally, change the appearance of the overlay chart.

To remove a chart overlay:

1. In the Charts window, click the **Overlays** button and, in the drop-down menu, select the chart that you want to remove.
2. Select **Delete Current** from the **Overlays** menu.

Changing the chart type of the Main chart

When you have selected the Main chart as the active chart, you can change the chart type also to types that do not support overlays (i.e., pie charts and gauge charts as well all types of 3d charts). In such a situation all overlays will be removed from the Charts window and XMLSpy displays a warning message.



5.8.10 Export

Clicking the **Export** button gives you the following options:

- *Save the chart as an image to file:* The image formats available are PNG, GIF, BMP, and JPG.
- *Copy the currently sized image or a resized to the clipboard:* Enables the chart to be subsequently copied from the clipboard into a report in another application.
- *Print chart:* Sends the image to a printer on your network. The height and width of the image can each be specified as a percentage of the page size.
- *Copy XSLT or XQuery code to the clipboard:* Creates an XSLT fragment or XQuery fragment. Each is essentially the Altova extension function `CreateChart`. This function can be used with other Altova extension functions and processed with XMLSpy or AltovaXML Engine to generate charts. To help you use the `CreateChart` extension

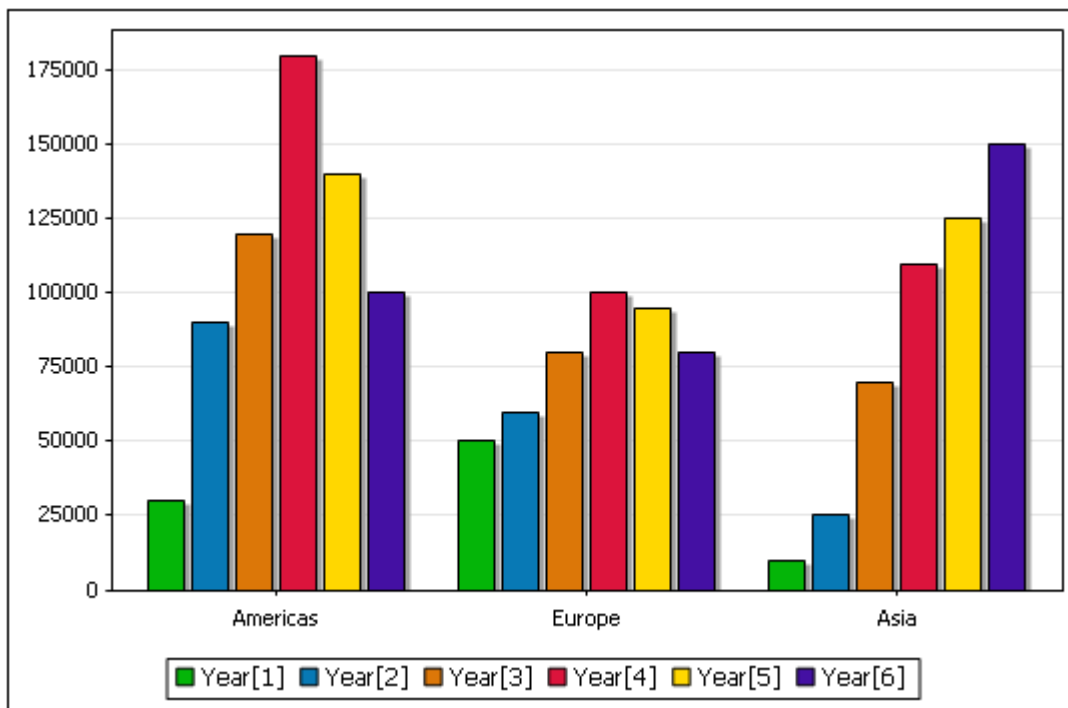
function, a commented-out usage example is also created with each fragment.

5.8.11 Chart Example: Simple

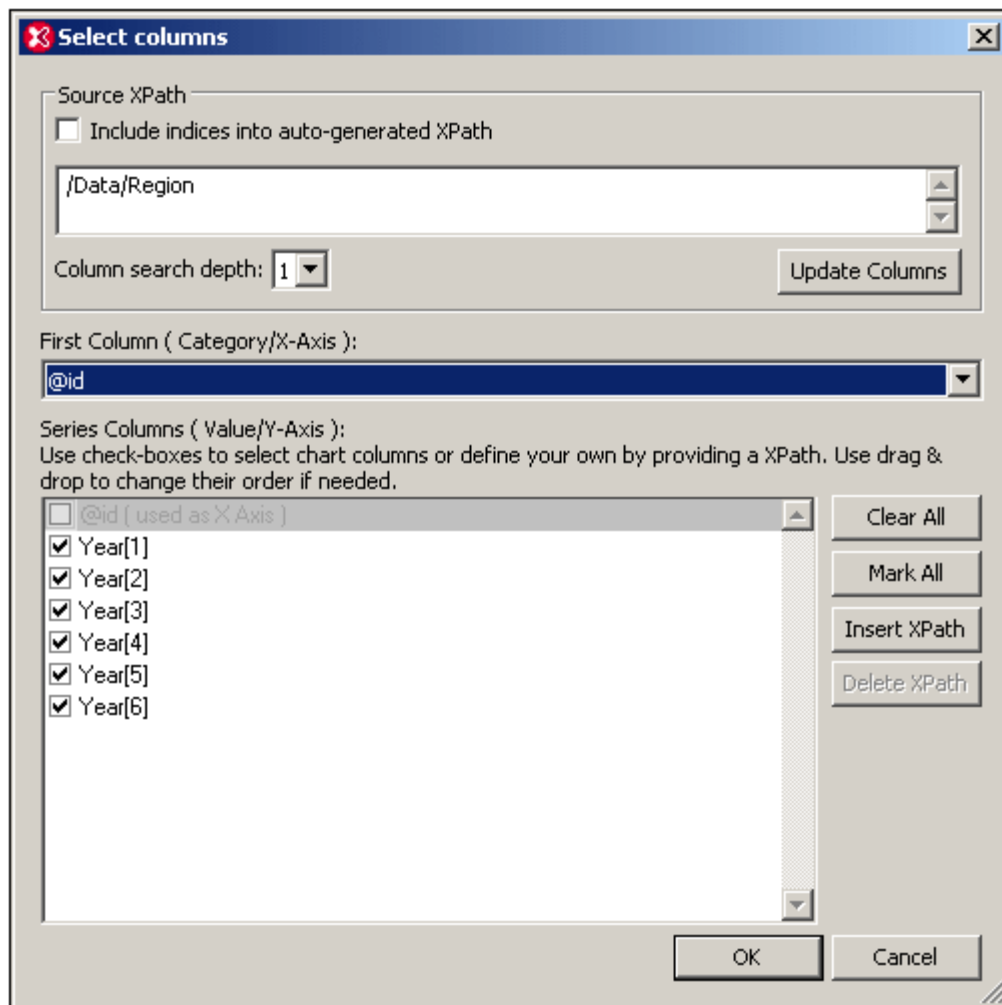
Consider the following XML document. (It is named `YearlySales.xml` and is available in the folder `C:\Documents and Settings\<username>\My Documents\Altova\XMLSpy2012\Examples\Tutorial.`)

```
<?xml version="1.0" encoding="UTF-8"?>
<Data xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="YearlySales.xsd">
  <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>
```

We wish to produce a chart that plots the three regions on the X-Axis and gives the yearly sales for each region. Our chart should look something like the bar chart below.



This is a simple chart to create because we can select the `Region` element as the Source XPath. The Source XPath expression returns a sequence of three items: the three `Region` elements. Each `Region` element will, in turn, be the context node for the X-Axis and Y-Axis data selections.



For the series we want the `Year` elements of each region, so a search depth of one level will suffice. We select the `Region` element's `id` attribute for the X-Axis. The `id` attribute values will therefore be used as the labels of the three X-Axis ticks. All the `Year` series are checked because we wish to include all the `Year` elements in the chart data table.

Clicking the **OK** button generates the chart we wanted. For more advanced charts, see the section, [Chart Example: Advanced](#).

5.8.12 Chart Example: Advanced

Consider the following XML document. (It is named `YearlySales.xml` and is available in the folder `C:\Documents and Settings\<username>\My Documents\Altova\XMLSpy2012\Examples\Tutorial.`)

```
<?xml version="1.0" encoding="UTF-8"?>
<Data xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance "
```

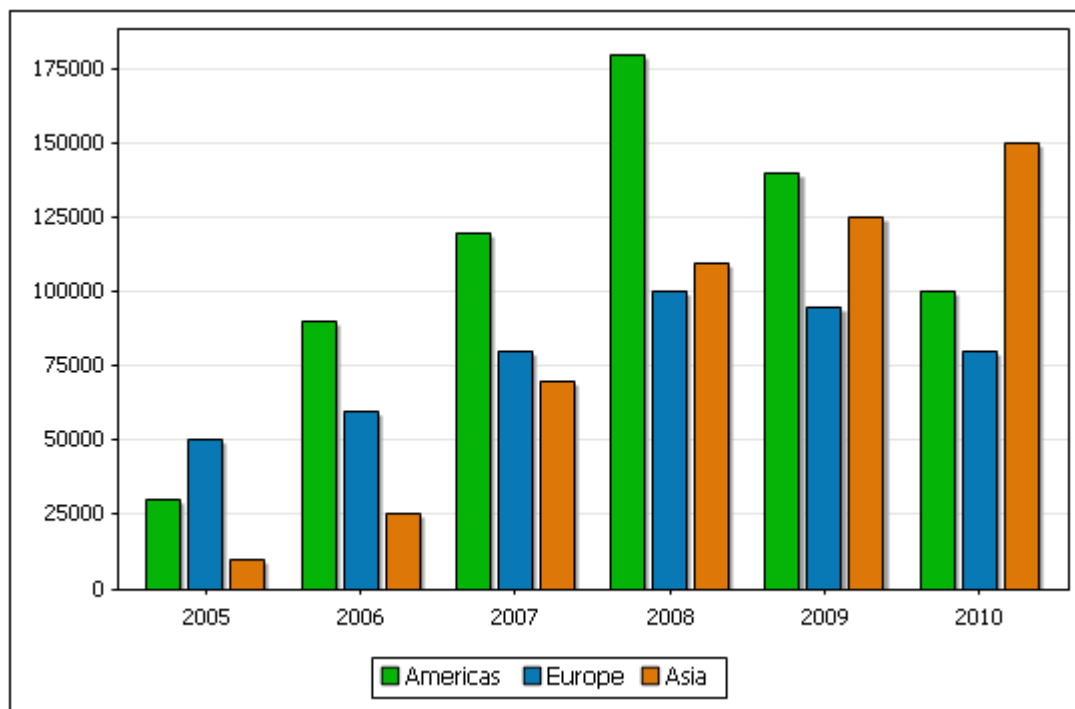


```

xsi:noNamespaceSchemaLocation="YearlySales.xsd">
  <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>

```

We wish to produce a chart that plots the years on the X-Axis and compare the regional sales for each year. Our chart should look something like the bar chart below.

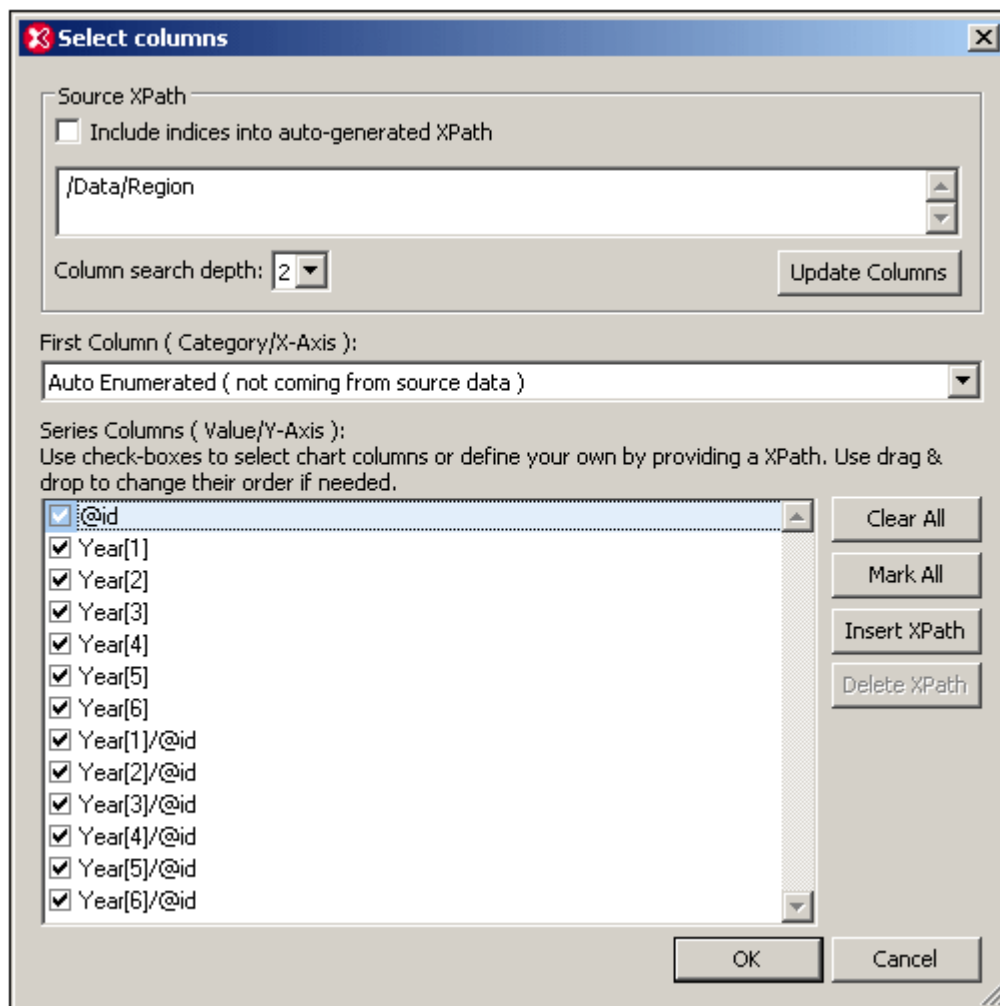


We show two ways in which this can be done. These two ways together demonstrate how the different data selection parameters can be combined to produce the required results.

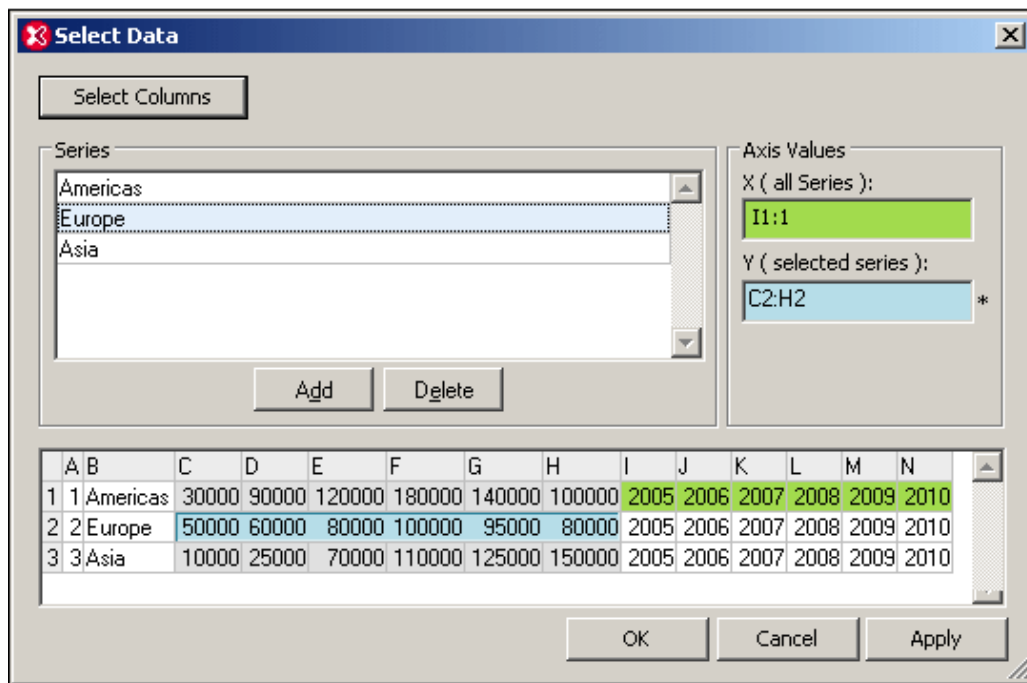
Method 1: Modifying Axis Values

In the first method, the axes that were selected in the Select Columns dialog are modified in the Select Data dialog. All that must be ensured is that all the required data is available for selection in the chart data table in the Select Data dialog.

1. In the Select Columns dialog, makes sure that all the required nodes will be available for X-Axis and Y-Axis selection. In the screenshot below, notice that the Column Search Depth has been set to 2 so that the `Year/@id` attributes are also selected.



2. In the Select Data dialog (*screenshot below*), the chart data table has the following columns: the first column is the X-Axis selection (which is the Auto-Enumerated selection), the remaining columns are the series (Y-Axis) columns, which are the `Region/@id` attributes, the `Year` element contents, and the `Year/@id` attributes. Notice also that: (i) there are only three rows, so three X-Axis ticks; (but we need six X-Axis ticks for the six years); (ii) there are 13 series columns.

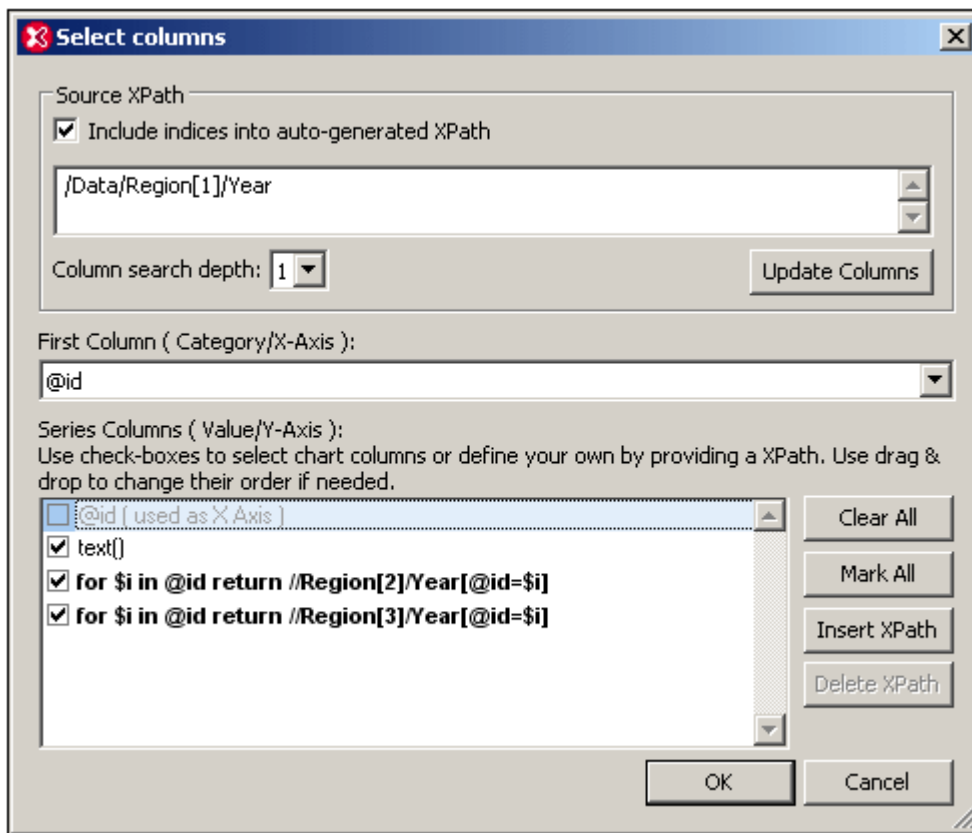


3. In the Series pane, we delete any 10 of the 13 series rows and rename the remaining three series to *Americas*, *Europe*, and *Asia*, as shown in the screenshot above. The order selected here will be the order of the X-Axis tick labeling.
4. In the Series pane, select the *Americas* series. In the Axis Values pane, click in the X-Axis box to enable modification. Then click the cell *I1* in the chart data table and drag to the cell *N1*. In the Y-Axis text box either enter *C1: H1* or make the selection by dragging from *C1* to *H1*.
5. For the *Europe* and *Asia* series, select *C2: H2* and *C3: H3*, respectively for the Y-Axis. The X-Axis selection can be the same as that for the *Americas* series.
6. Click **OK**. The required chart is generated.

Note: The number of X-Axis ticks (defined by default by the number of rows in the chart data table) is increased from three to six because the number of X-Axis labels is six.

Method 2: Generating series with XPath expressions

In the second method, XPath expressions are inserted to generate series. This is necessary because the Source XPath (see *screenshot below*) does not have as its descendants the nodes wanted for the series. However, the Source XPath does generate six X-Axis ticks (by selecting the *Year* elements of the first *Region* element). In order for the first *Region* element to be selected using the *[1]* predicate, the *Include Indices* check box must be checked and the **Update Columns** button clicked.

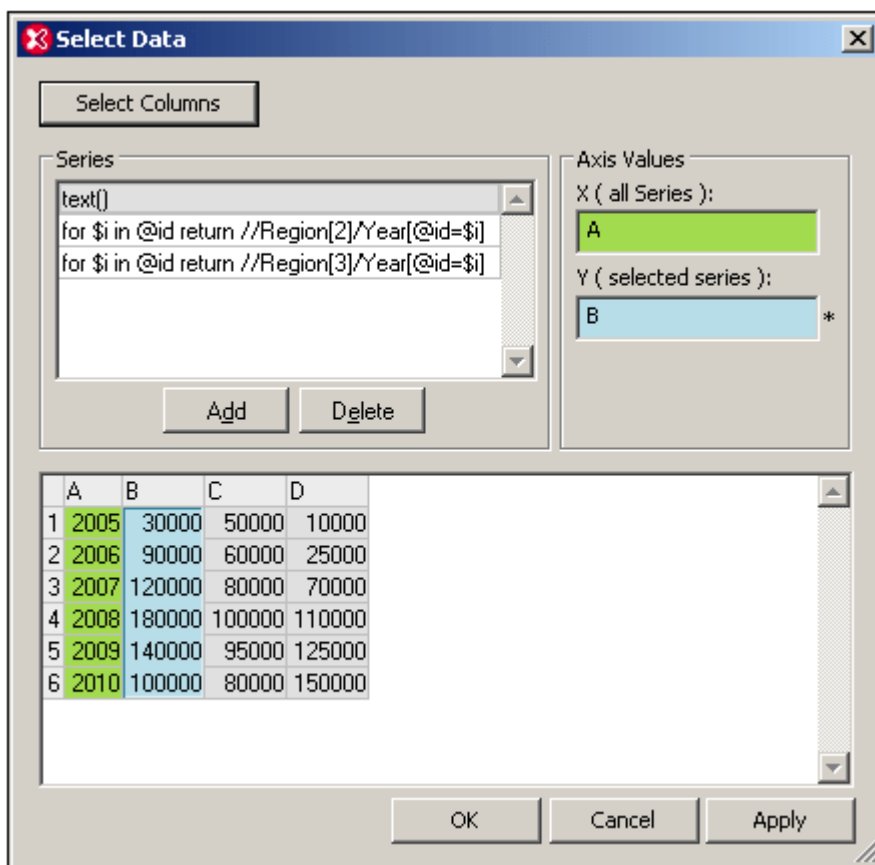


In the Select Columns dialog, the `Region[1]/Year` element has only two descendants: `@id` and `text()`. The `@id` attribute is selected for the X-Axis, thereby generating the correct X-Axis label for each of the six X-Axis ticks. The chart data table would be evaluated as follows.

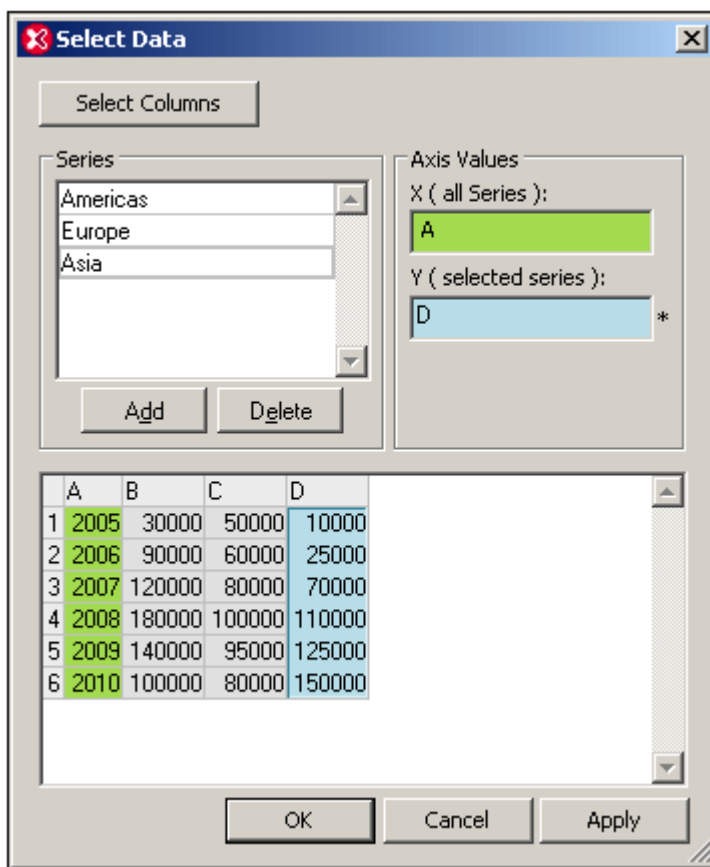
Source XPath	X-Axis	Y-Axis (Series columns)		
<code>Region[1]/Year[1]</code>	<code>@id</code>	<code>text()</code>	XPath-1	XPath-2
<code>Region[1]/Year[2]</code>	<code>@id</code>	<code>text()</code>	XPath-1	XPath-2
<code>Region[1]/Year[3]</code>	<code>@id</code>	<code>text()</code>	XPath-1	XPath-2
<code>Region[1]/Year[4]</code>	<code>@id</code>	<code>text()</code>	XPath-1	XPath-2
<code>Region[1]/Year[5]</code>	<code>@id</code>	<code>text()</code>	XPath-1	XPath-2
<code>Region[1]/Year[6]</code>	<code>@id</code>	<code>text()</code>	XPath-1	XPath-2

Note that the context node is each of the six `Region[1]/Year` elements in turn. The first XPath expression looks for the current `Year/@id` attribute value and returns the `Region[2]/Year` element that has the same `Year/@id` value as the `@id` value of the current `Region[1]/Year`. The second XPath expression does the same for the `Region[3]/Year` elements. In this way, for each of the six years: the three Y-Axis series are the `Year` element children, respectively, of each of the three `Region` elements. (The `text()` node returns the contents of the `Region[1]/Year` elements.)

The chart data table in the Select Data dialog would look something like this.



The names of the series in the Select Data dialog can be changed from XPath expressions (as in the screenshot above) into meaningful legends (screenshot below). For each series the correct data column can be assigned in the Axis Values pane (by clicking in the Y-Axis text box and then selecting the required column in the chart data table).



Both the methods shown above generate identical charts. The different approaches are intended to show how the data selection parameters are to be used.

5.8.13 Chart Example: Candlestick

Candlestick charts are typically used for representing the movement of share prices on the stockmarket. There are two types of candlestick charts:

- Four-series candlestick charts, representing the opening, the highest, the lowest, and the closing prices of the day.
- Three-series candlestick charts, representing the the highest, the lowest, and the closing prices of the day.

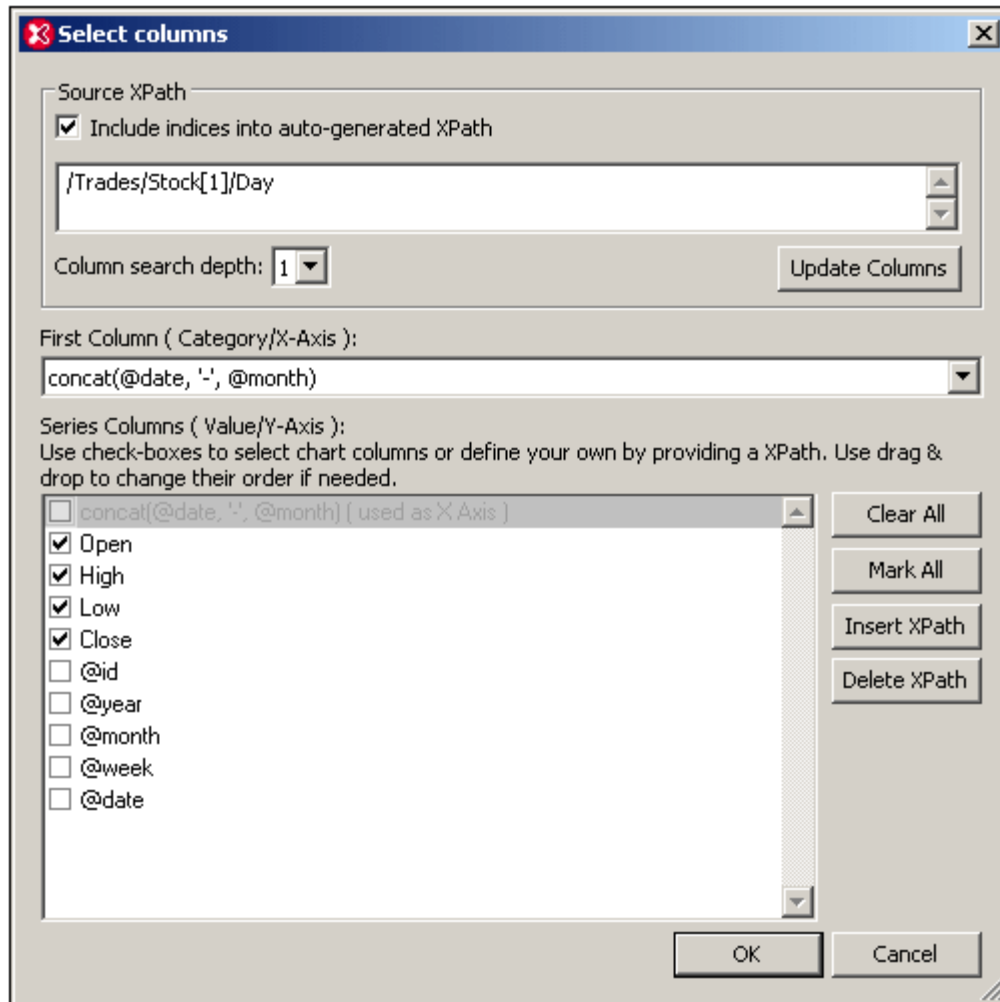
The file `Candlestick.xml`, which is available in the folder `C:\Documents and Settings\<username>\My Documents\Altova\XMLSpy2012\Examples\Tutorial`, is an example of an XML document structure that could be used for candlestick charts. The listing below shows the essential structure of the file.

```
<Trades>
  <Stock name="MyStock">
    <Day id="20110103" year="2011" month="Jan" week="01" date="03">
      <Open>90</Open>
      <High>110</High>
      <Low>88</Low>
      <Close>105</Close>
    </Day>
    ....
  </Stock>
</Trades>
```

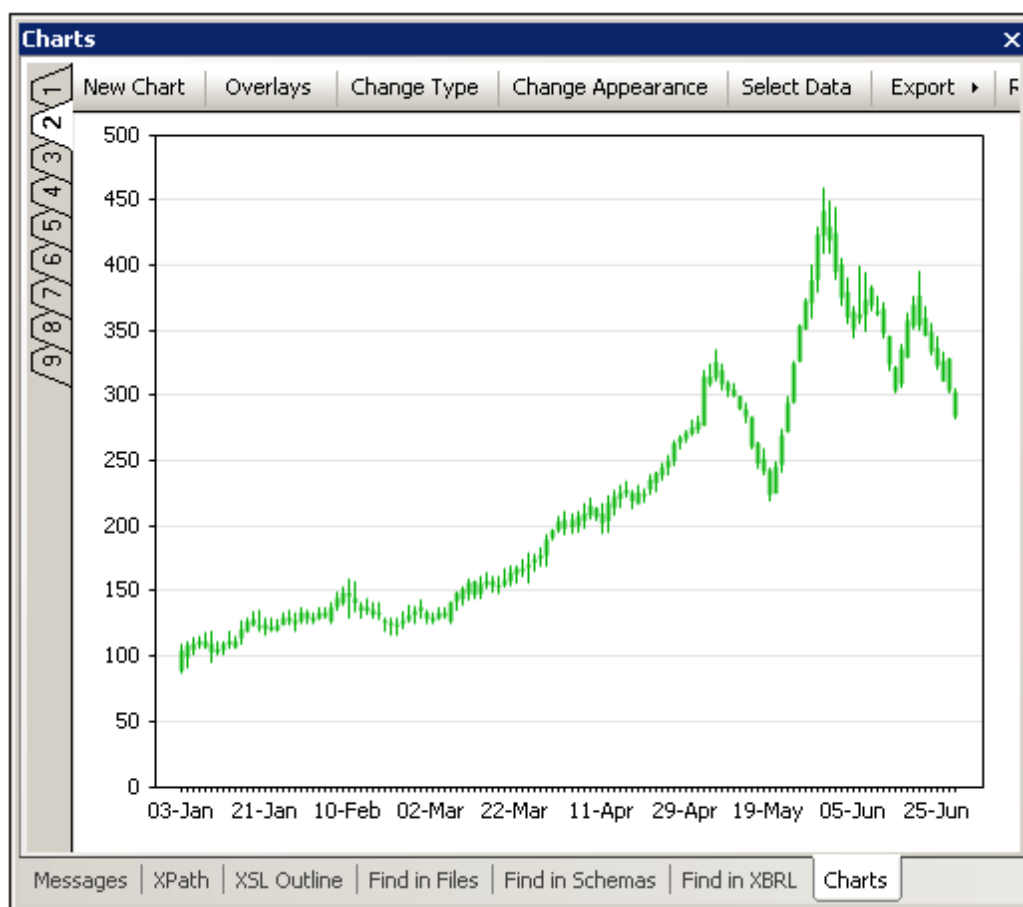
</Trades>

A candlestick chart can be created for the file mentioned above as follows:

1. With the cursor inside the Day element tag, click the **New Chart** button of the [Charts window](#). This pops up the Select Columns dialog (*screenshot below*). If the Include Indices check box is not checked, then check it and click **Update Columns**.



2. Click the Insert XPath button and insert the XPath expression: `concat(@date, '-', @month)`.
3. From the dropdown list of the First Column combo box, select the XPath expression you just entered. This will create the date and month of each `Day` element as the labels of the X-Axis ticks.
4. For the Y-Axis series, select the Open, High, Low, and Close check boxes in the Series Columns pane.
5. Click **OK**.
6. Click the **Change Type** button of the Charts window and change the chart type to Candlestick.
7. Click **OK**. This will create a candlestick chart like the one in the screenshot below.



Selecting a data subset

If you wish to view a subset of the chart data selected in the Select Columns dialog, for example, if you wish to view a certain range of dates within the chart data selection, click the **Select Data** button of the Charts window. This pops up the Select Data dialog (*screenshot below*).

Select Data

Select Columns

Series

- Open
- High
- Low
- Close

Add Delete

Axis Values

X (all Series):

A80:A127

Y (selected series):

B80:B127 *

Candlestick charts require 3 or 4 series of values:
 3 series: high, low, close
 4 series: open, high, low, close

	A	B	C	D	E
1	03-Jan	90	110	88	105
2	04-Jan	100	112	92	109
3	05-Jan	106	115	102	110
4	06-Jan	110	116	107	112
5	07-Jan	112	119	107	108
6	10-Jan	110	120	96	104
7	11-Jan	104	112	102	106
8	12-Jan	106	112	102	110
9	13-Jan	110	120	107	112

OK Cancel Apply

In the Axis Values pane, enter the X-Axis range, for example, the cells A80: A127, as in the screenshot above. For the various series, first click in the Y-Axis text box, then select the series in the Series pane, and then enter the range for that series. Do this for each of the four series. For example, the screenshot above shows the Y-Axis range selected for the Open series. For more information about the Select Data dialog, see the section, [Chart Data](#).

5.9 XML Signatures

An XML file can be digitally signed and the signature can be subsequently verified. If the file has been changed after it was signed, then the verification will fail. XMLSpy supports both the creation and the verification of XML signatures.

XML signatures in XMLSpy views

XML signatures can be created for all types of XML files, including for XML Schema, WSDL, and XBRL files. The [XML | Create XML Signature](#) and [XML | Verify XML Signature](#) commands, therefore, are available in all XMLSpy views: [Text View](#), [Grid View](#), [Schema View](#), [WSDL View](#), and [XBRL View](#).

How XML signatures work

The entire process, from signature-creation to signature-verification, works as follows:

1. The XML file is signed using either the private key of a certificate or a password. In XMLSpy you can create a signature using the [XML | Create XML Signature](#) command. The signature is obtained by processing: (i) the XML document, and (ii) the private key of a certificate, or a password.
2. The signature can be either included with the XML file or stored in a separate file.
3. The signature of the XML file is verified using the public key of the certificate or the password. The verification process works by, first, processing: (i) the XML document, and (ii) the public key of the certificate, or the password, used for signing, and, second, comparing this result with the signature. If the XML file was changed after it was signed, then the verification will fail. In XMLSpy you can verify a signature using the [XML | Verify XML Signature](#) command.

The details of how to create and verify signatures in XMLSpy are described in sub-sections of this section:

- [Creating XML Signatures](#)
- [Verifying XML Signatures](#)

How certificates are used in XML signatures

To be used with XML signatures, certificates must have a private key and public key. The private key is used to create the XML signature, the public key is used to verify the XML signature.

In a typical scenario, the sender of an XML document has access to the private key of a certificate and creates the XML signature with it. The receiver of the document will have access to the public key of the certificate. This access can be of two types: (i) The sender sends the public key information with the signature; (ii) The receiver has access to a public-key version of the certificate used by the sender.

For more details about certificates, see the sub-section, [Working with Certificates](#).

Note: XMLSpy's XML Signature feature supports only certificates of type RSA-SHA1 and DSA-SHA1.

XML document validity

If an XML signature is embedded in the XML document, a `Signature` element in the

namespace `http://www.w3.org/2000/09/xmldsig#` is added to the XML document. In order for the document to remain valid according to a schema, the schema must contain the appropriate element declarations. XMLSpy embeds signatures in two ways:

- *Enveloped*: The `Signature` element is created as the last child element of the root (or document) element.
- *Enveloping*: The `Signature` element is created as the root (or document) element, and the original XML document element is placed inside a child element of the signature element named `Object`.

If you do not wish to modify the schema of the XML document, the XML signature can be created in an external file. For more details, see the description of the placement options in the section, [Creating XML Signatures](#).

Given below are excerpts from XML Schemas that show how the `Signature` element of an enveloped signature can be allowed. You can use these examples as guides to modify your own schemas.

In the first of the two listings below, the XML Signature Schema is imported into the user's schema. The XML Signature Schema is located at the web address:

<http://www.w3.org/TR/xmldsig-core/xmldsig-core-schema.xsd>

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsig="http://www.w3.org/2000/09/xmldsig#"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  <xs:import namespace="http://www.w3.org/2000/09/xmldsig#"
    schemaLocation="
http://www.w3.org/TR/xmldsig-core/xmldsig-core-schema.xsd" />
  <xs:element name="Root">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="FirstChildOfRoot"/>
        <xs:element ref="SecondChildOfRoot" minOccurs="0"/>
        <xs:element ref="ThirdChildOfRoot" minOccurs="0"/>
        <xs:element ref="xsig:Signature" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  ...
</xs:schema>
```

A second option (*listing below*) is to add a generic wildcard element which matches any element from other namespaces. Setting the `processContents` attribute to `lax` causes the validator to skip over this element—because no matching element declaration is found. Consequently, the user does not need to reference the XML Signatures Schema. The drawback of this option, however, is that any element (not just the `Signature` element) can be added at the specified location in the XML document without invalidating the XML document.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">
  <xs:element name="Root">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="selection"/>
        <xs:element ref="newsitems" minOccurs="0"/>
        <xs:element ref="team" minOccurs="0"/>
        <xs:any namespace="##other" minOccurs="0" processContents="lax"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  ...
</xs:schema>
```

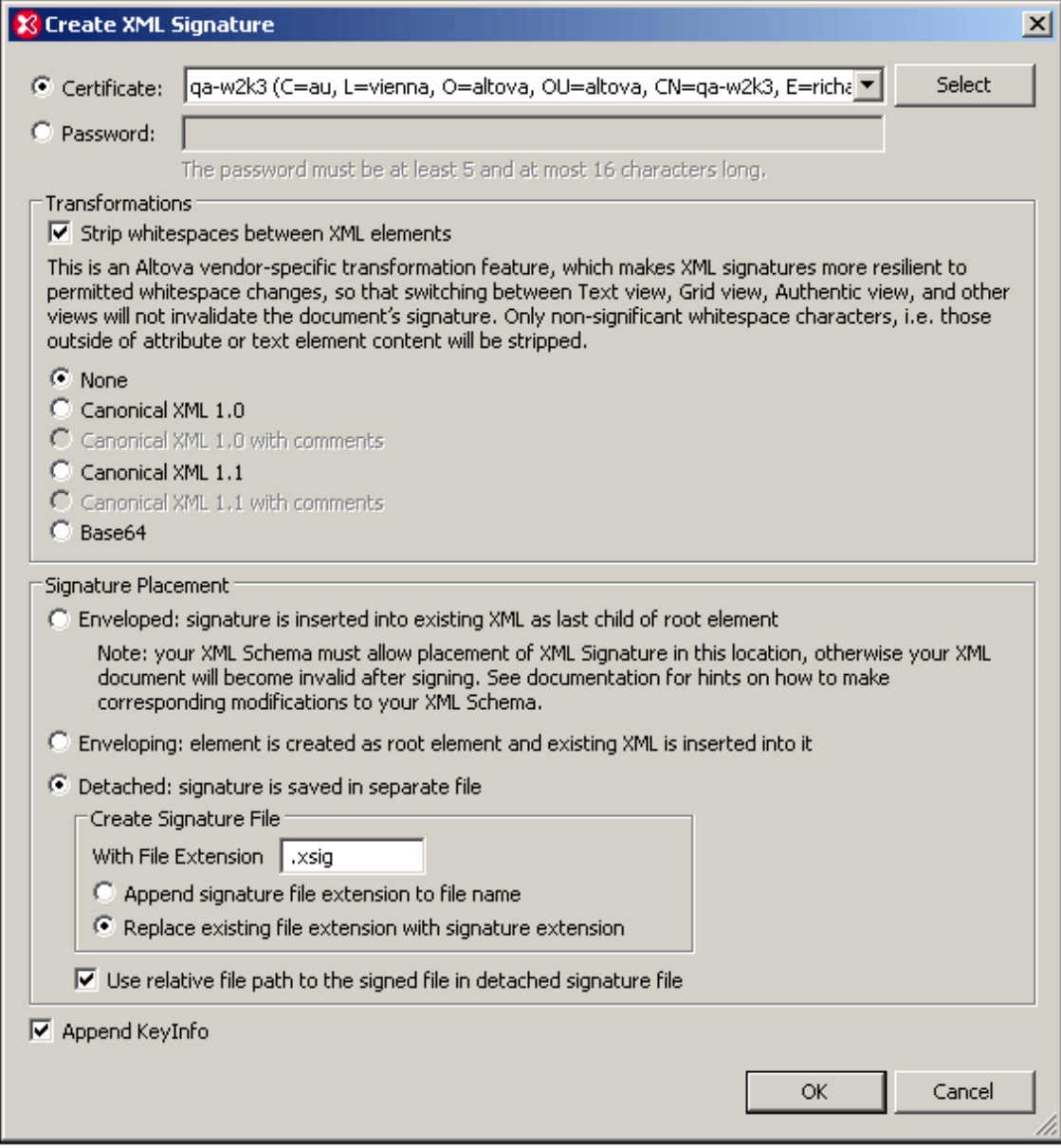
```
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    ...
  </xs:schema>
```

W3C Specification

For more details about XML signatures, see the W3C specification for XML signatures at <http://www.w3.org/TR/xmldsig-core/>.

5.9.1 Creating XML Signatures

To create an XML signature for an XML document, open the XML document for which you wish to create a signature. Then click the menu command **XML | Create XML Signature**. This opens the Create XML Signature dialog (*screenshot below*), the settings of which are explained below.



The dialog box "Create XML Signature" contains the following sections:

- Certificate/Password:** Radio buttons for "Certificate" and "Password". The "Certificate" section has a dropdown menu showing "qa-w2k3 (C=au, L=vienna, O=altova, OU=altova, CN=qa-w2k3, E=richa..." and a "Select" button. The "Password" section has a text input field and a note: "The password must be at least 5 and at most 16 characters long."
- Transformations:** A checkbox "Strip whitespaces between XML elements" is checked. Below it is a text description: "This is an Altova vendor-specific transformation feature, which makes XML signatures more resilient to permitted whitespace changes, so that switching between Text view, Grid view, Authentic view, and other views will not invalidate the document's signature. Only non-significant whitespace characters, i.e. those outside of attribute or text element content will be stripped." Below this are radio buttons for "None", "Canonical XML 1.0", "Canonical XML 1.0 with comments", "Canonical XML 1.1", "Canonical XML 1.1 with comments", and "Base64".
- Signature Placement:** Radio buttons for "Enveloped: signature is inserted into existing XML as last child of root element", "Enveloping: element is created as root element and existing XML is inserted into it", and "Detached: signature is saved in separate file". The "Detached" option is selected. Below it is a sub-section "Create Signature File" with a text input field "With File Extension" containing ".xsig", and two radio buttons: "Append signature file extension to file name" and "Replace existing file extension with signature extension" (which is selected). Below this is a checkbox "Use relative file path to the signed file in detached signature file" which is checked.
- Append KeyInfo:** A checkbox "Append KeyInfo" is checked.
- Buttons:** "OK" and "Cancel" buttons at the bottom right.

Authentication method: certificate or password

The signature can be based on a certificate or a password. Select the radio button of the method you wish to use.

- **Certificate:** Click the **Select** button and browse for the certificate you want. The certificate you select must have a private key. The signature is generated using the private key of the certificate. To verify the signature, access to the certificate (or to a public-key version of it) is required. The public key of the certificate is used to verify the signature. For more details about certificates, see the section [Working with Certificates](#).
- **Password:** Enter a password with a length of five to 16 characters. This password will subsequently be required to verify the signature.

Transformations

The XML data is transformed and the result of the transformation is used for the creation of the signature. You can specify the canonicalization algorithm to be applied to the file's XML data (the `SignedInfo` content) prior to performing signature calculations. Significant points of difference between the algorithms are noted below:

- *Canonical XML with or without comments*: If comments are included for signature calculation, then any change to comments in the XML data will result in verification failure. Otherwise, comments may be modified or be added to the XML document after the document has been signed, and the signature will still be verified as authentic.
- *Base64*: The root (or document) element of the XML document is considered to be Base64 encoded, and is read in its binary form. If the root element is not Base64, an error is returned or the element is read as empty.
- *None*: No transformation is carried out and the XML data from the binary file saved on disk is passed directly for signature creation. Any subsequent change in the data will result in a failed verification of the signature. However, if the *Strip Whitespace* check box option is selected, then all whitespace is stripped and changes in whitespace will be ignored. A major difference between the *None* option and a *Canonicalization* option is that canonicalization produces an XML data stream, in which some differences, such as attribute order, are normalized. As a result, a canonicalization transformation will normalize any changes such as that of attribute order (so verification will succeed), while no-transformation will reflect such a change (verification will fail). Note, however, that a default canonicalization is performed if the signature is embedded (enveloped or enveloping). So the XML data will be used as is (i.e. with no transformation), when the signature is detached, *None* is selected, and the *Strip Whitespaces* checkbox is unchecked.

Signature placement

The signature can be placed within the XML file or be created as a separate file. The following options are available:

- *Enveloped*: The `Signature` element is created as the last child element of the root (document) element.
- *Enveloping*: The `Signature` element is created as the root (document) element and the XML document is inserted as a child element.
- *Detached*: The XML signature is created as a separate file. In this case, you can specify the file extension of the signature file and whether the file name is created with: (i) the extension appended to the name of the XML file (for example, `test.xml.xsig`), or (ii) the extension replacing the XML extension of the XML file (for example, `test.xsig`). You can also specify whether, in the signature file, the reference to the XML file is a relative or an absolute path.

Note: XML signatures for XML Schema (.xsd) files can be created from Schema View as detached signature files (not embedded). XML signatures for XBRL files can be created from XBRL View as detached signature files (not embedded). XML signatures for WSDL files can be created from WSDL View as detached signature files, or they can be "enveloped" in the WSDL file.

Note: If the XML signature is created as a detached (separate) file, then the XML file and signature file are associated with each other via a reference in the signature file. Consequently, signature verification in cases where the signature is in an external file must be done with the signature file active—not with the XML file active.

Append key information

The *Append Keyinfo* option is available when the signature is certificate-based. It is unavailable if the signature is password-based.

If the option is selected, public-key information is placed inside the signature, otherwise key information is not included in the signature. The advantage of including key information is that the certificate itself (specifically the public-key information in it) will not be required for the verification process (since the key information is present in the signature).

5.9.2 Verifying XML Signatures

An XML signature will be correctly verified if the XML file has not been changed since having been signed. Otherwise the verification will fail. XML signatures can be verified in XMLSpy in the following circumstances as described below:

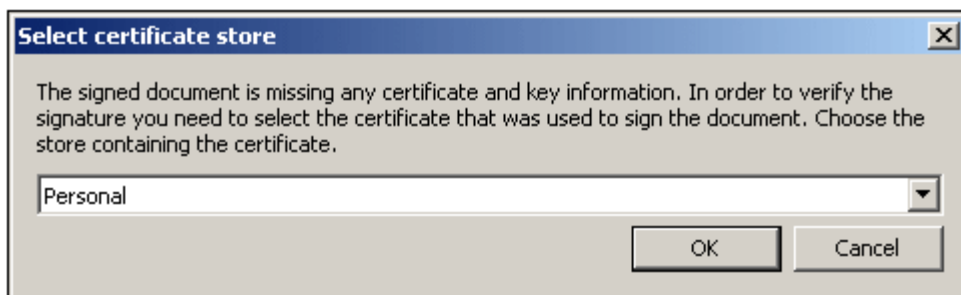
- [XML file contains certificate-based signature, key information included in signature](#)
- [XML file contains certificate-based signature, key information not contained in signature](#)
- [Certificate-based signature in external file, key information contained in signature](#)
- [Certificate-based signature in external file, key information not contained in signature](#)
- [XML file contains password-based signature](#)
- [Password-based signature in external file](#)

XML file contains certificate-based signature, key information included in signature

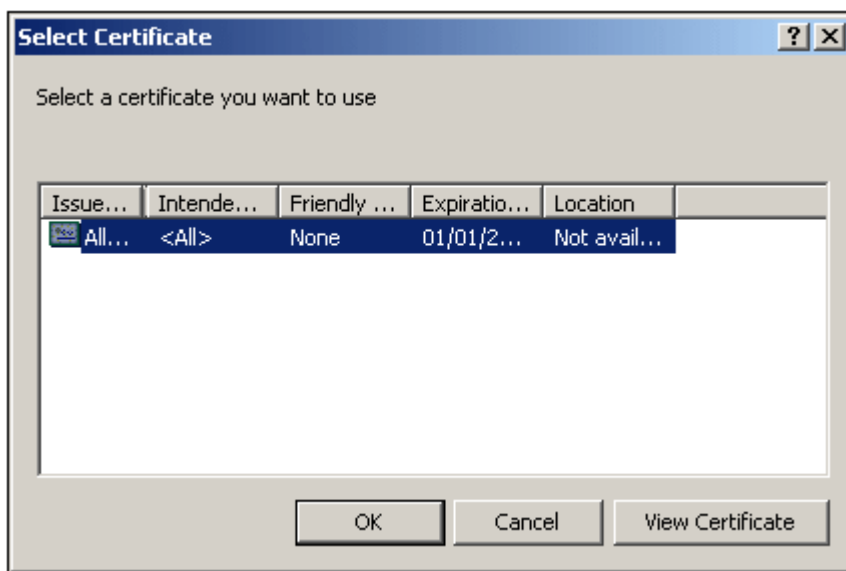
To verify the XML signature in this scenario, make the XML file active in XMLSpy. On clicking the **XML | Verify XML Signature** command, the verification process will be executed and the result will be displayed in the Messages window (verification succeeded or failed).

XML file contains certificate-based signature, key information not contained in signature

If no key information is contained in the certificate-based signature, XMLSpy will prompt you for the certificate from which public-key information for the verification can be read. Verification is done with the XML file active in XMLSpy. On clicking the **XML | Verify XML Signature** command, you will be prompted to select the [certificate store](#) in which the certificate is stored (*screenshot below*).



On selecting a [certificate store](#) and clicking **OK**, a dialog displaying the certificates in that store pops up (*screenshot below*). Select the certificate required for the verification and click **OK**.



The verification process is executed and the result is displayed in the Messages window.

Certificate-based signature in external file, key information contained in signature

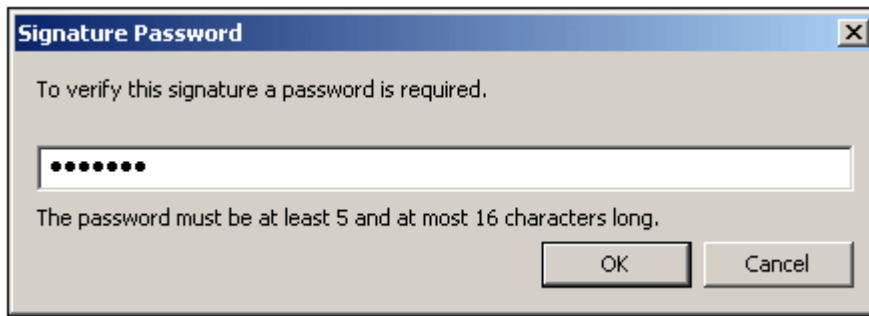
If a certificate-based XML signature is in an external file, the signature is verified with the signature file active in XMLSpy. On clicking the **XML | Verify XML Signature** command, the verification process will be executed and the result will be displayed in the Messages window (verification succeeded or failed).

Certificate-based signature in external file, key information not contained in signature

If a certificate-based XML signature is in an external file, the signature is verified with the signature file active in XMLSpy. On clicking the **XML | Verify XML Signature** command, XMLSpy will prompt you for the certificate from which public-key information for the verification can be read. Select the certificate as described in the section: [XML file contains certificate-based signature, key information not contained in signature](#). The verification process will be executed and the result will be displayed in the Messages window (verification succeeded or failed).

XML file contains password-based signature

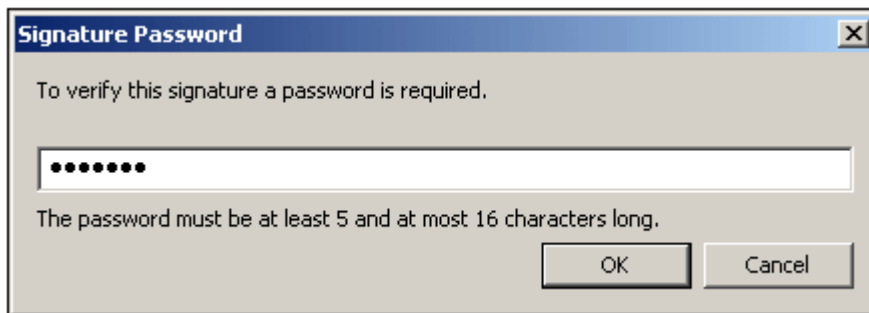
If the XML file contains a password-based XML signature, the signature is verified with the XML file active in XMLSpy. On clicking the **XML | Verify XML Signature** command, a dialog pops up prompting you for the password (*screenshot below*).



Enter the password, which must be five to sixteen characters long, and then click **OK**. The verification process will be executed and the result will be displayed in the Messages window (verification succeeded or failed).

Password-based signature in external file

If a password-based XML signature is in an external file, the signature is verified with the signature file active in XMLSpy. On clicking the **XML | Verify XML Signature** command, a dialog pops up prompting you for the password (*screenshot below*).



Enter the password, which must be five to sixteen characters long, and then click **OK**. The verification process will be executed and the result will be displayed in the Messages window (verification succeeded or failed).

5.9.3 Working with Certificates

Authorization certificates are commonly used to create and verify XML signatures. This section contains information about obtaining, importing, and exporting certificates. It is organized into the following sub-sections:

- [Obtaining a certificate with a private-public-key pair](#)
- [Importing a private-public-key certificate](#)
- [The certificate stores on a Windows machine](#)
- [Exporting a public-key certificate](#)

Obtaining a certificate with a private-public-key pair

A certificate can be obtained in the following ways:

- *From a certificate authority.* The certificate authority verifies the identity of the certificate's owner. Certificates obtained in this way are in contrast to self-signed certificates, which can be created by anyone with a certificate creation tool.
- *By creating a self-signed certificate.* Such certificates are not verified by any authority,

but often provide adequate security. A number of certificate creation tools, such as Microsoft's Visual Studio, are available.

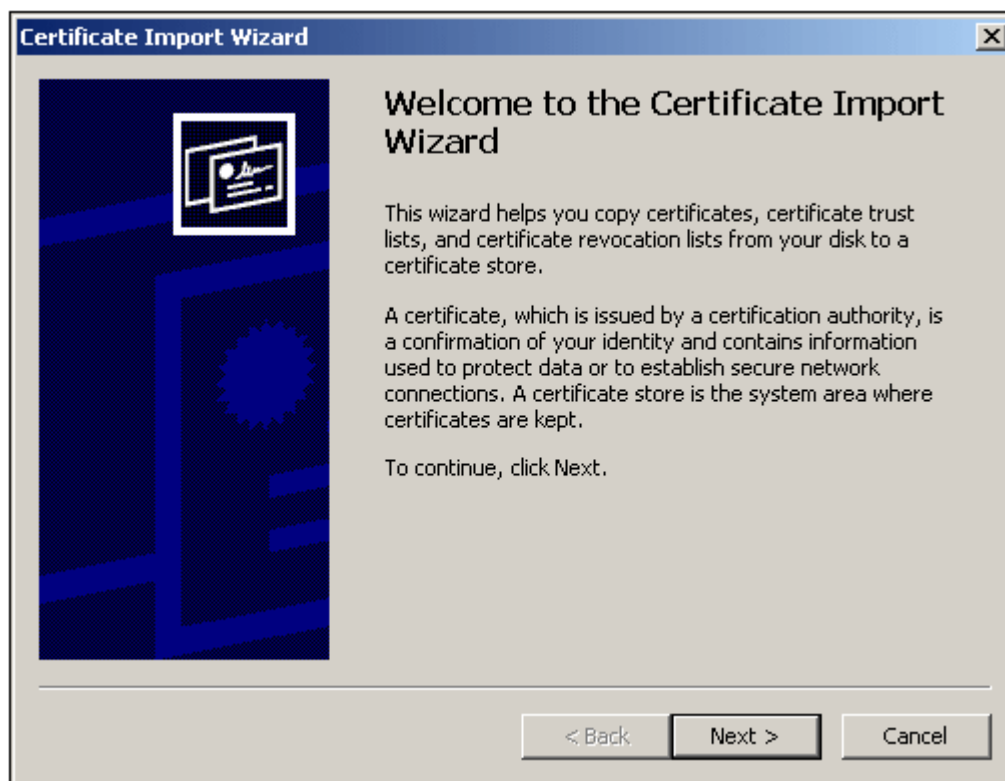
For use with XML signatures you will need a certificate with a private-public-key pair.

Note: XMLSpy's XML Signature feature supports only certificates of type RSA-SHA1 and DSA-SHA1.

Importing a private-public-key certificate

After a private-public-key certificate has been obtained, you will need to import it to your Windows certificate store. Do this as follows:

1. Double-click the certificate file to open the Certificate Import Wizard (*screenshot below*), and click **Next**.

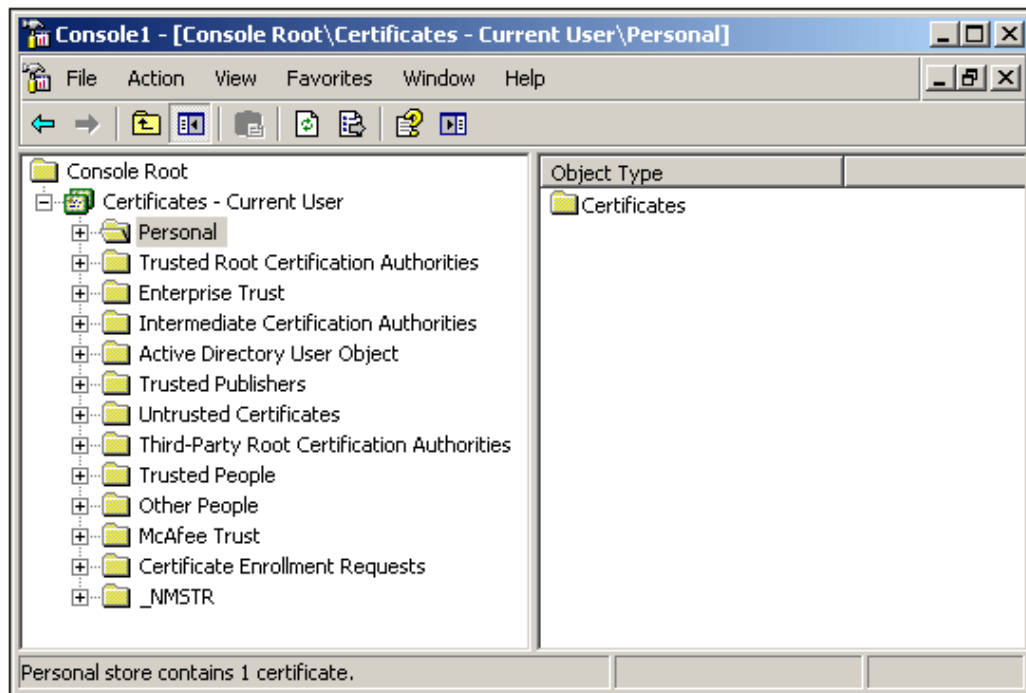


2. In the File to Import window, ensure that the certificate file is selected, then click **Next**.
3. Type in the password for the private key. You must know the password if you intend to use the private key to create an XML signature. The password for the private key will be supplied to you when you obtain the certificate. After typing in the password, click **Next**.
4. You can allow the wizard to automatically select the store in which to place the certificate—according to the certificate type—or you can select the store yourself. (It might be better to select the store yourself, so you know the location of the certificate.) Click **Next** when done.
5. Click **Finish** to complete the process.

Certificate stores on a Windows machine

The certificate store on a Windows XP machine can be accessed as follows:

1. In the Start menu, select **Run**.
2. Type in `mmc` and click **OK**. A Console window pops up (*screenshot below*).



3. In the Console window, select the command **File | Add/Remove Snap-in**.
4. In the *Standalone* tab of the Add/Remove Snap-in dialog that pops up, click **Add**.
5. In the Add Standalone Snap-in dialog that pops up, select **Certificates** and click **Add**.
6. Close the Add Standalone Snap-in dialog.
7. In the Add/Remove Snap-in dialog, click **OK**.
8. The Console Root in the Console window will now contain a **Certificates** item (see *screenshot above*). This **Certificates** item contains the certificate stores of your machine.
9. Save the Console as a Microsoft Management Console File (`.msc` file) via the **File | Save** command of the Console window. You can subsequently use this MSC file (via the **File | Open** command of a Console window) to access the certificate stores on your machine.

Exporting a public-key certificate

If you have a certificate with a private-public key, you might wish to export this certificate with only a public key. This public-key certificate can then be sent to receivers for use in verifying signatures created with the private key of the certificate.

A public-key certificate can be exported from an existing private-public-key certificate as follows:

1. Open the certificate stores in a Console window. Do this as follows: (i) Enter `mmc` on the **Start** menu's **Run** command line; (ii) In the Console window that pops up, select **File | Open**, and select the MSC file in which the certificate stores were saved (see *section immediately above this section*).
2. Browse for the certificate that you wish to export as a public-key certificate and right-click it.
3. In the context menu that pops up, select **All Tasks | Export**. This pops up the Certificate Export Wizard (*screenshot below*).



4. Select **Next**.
5. In the Export Private Key window, select *No, do not export the private key*, and click **Next**.
6. In the Export File Format window, select the required format (leave the default DER format unchanged if you are not sure), and click **Next**.
7. In the File to Export window, browse for the location where you wish to save the file and provide a name for the file (without a file extension, which will be automatically appended). Click **Next** when done.
8. Click **Finish** to complete the export.

A public-key certificate will be created at the location you specified. This public-key certificate can be sent to receivers of XML files signed with the corresponding private key. The receiver can then import this public-key certificate to a certificate store on his or her machine and use the public key of this certificate for verification.

5.10 Additional Features

Additional features for working with XML files are listed below.

- [Encoding](#)
- [Generating DTDs and XML Schemas](#)
- [Find and Replace](#)
- [Evaluating XPath](#)
- [Importing and exporting text](#)

Encoding

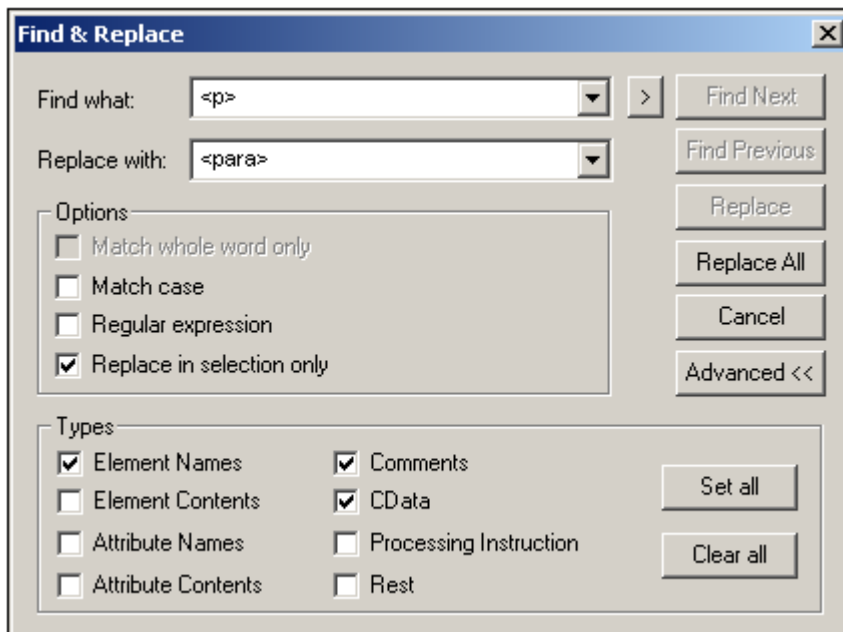
The encoding of XML files (and other types of documents) can be set via the menu command [File | Encoding](#). The default encoding of XML and non-XML files can be specified in the [Options | Encoding](#) tab.

Generating DTDs and XML Schemas

If you wish to create a schema that describes the structure of an XML document, use the [DTD/Schema | Generate DTD/Schema](#) menu command. In the Generate DTD/Schema dialog that appears, you can select whether to generate a DTD or an XML Schema as well as certain XML Schema options, such as whether to generate enumerations from the values contained in the XML document.

Find and Replace

The [Find](#) and [Replace](#) features (accessed via the **Edit** menu) has powerful search capabilities. The search term can be defined additionally in terms of casing and whether whole words should be matched, and it can also be expressed as a regular expression. The search range can be restricted to a selection in the document and to particular node types (*see screenshot below*).



The screenshot above shows the Find & Replace dialog in Text View. The dialog and functionality varies according to the view that is active.

Evaluate XPath

An XPath expression, which you enter in the XPath Window, can be evaluated against the active XML document. The results of the evaluation are displayed in the XPath Window, and clicking a node in the result highlights that node in the document display in the Main Window. Note that the XPath Window can be made active by clicking **XML | Evaluate XPath** command.

Importing and exporting text

Text data can be imported from, and exported to, other application formats. Commands for these features are in the [Convert](#) menu.

6 DTDs and XML Schemas

Altova website:  [XML Schema Editor](#)

This section provides an overview of how to work with [DTDs](#) and [XML Schemas](#). It also describes [SchemaAgent](#) and the powerful [Find in Schemas](#) feature. In addition to the editing features, XMLSpy provides the following powerful DTD/Schema features:

Catalog mechanism

Support for the OASIS [catalog mechanism](#) enables the re-direction of URIs to local addresses, thus facilitating use across multiple workstations.

Schema rules

An XML Schema can be assigned a set of additional constraints defined by the user. XMLSpy contains a Schema Rule Editor in which a Schema Rule Set for an XML Schema can be created and edited.

Schema subsets

Components of a large schema can, in Schema View, be created as a separate file. These smaller schema subsets can then be included in the larger schema. The reverse operation, known as flattening a schema, puts the components of included files directly in the larger schema. How to generate schema subsets and flatten schemas is described in the section, [Schema Subsets](#).

Converting DTDs to XMLSchemas and vice versa

A DTD can be converted to an XML Schema and vice versa via the [DTD/Schema | Convert DTD/Schema](#) menu command. See the [description of this file command](#) in the User Reference section for a description of how to use it.

Generate Sample XML file

You can generate, via the [DTD/Schema | Generate Sample XML File](#) menu command, a skeleton XML document based on the active DTD or XML Schema file. This is very useful for quickly creating an XML file based on a schema.

Go to definition

When the cursor is located within a node in an XML document, clicking the [DTD/Schema | Go to Definition](#) menu command opens the schema file and highlights the definition of the selected XML node.

- *Generate sample XML file from DTD:* With the [DTD/Schema | Generate Sample XML File](#) command, an XML document can be generated that is based on the active DTD.

6.2 XML Schemas

XML Schema documents can be edited in Text View, Grid View, and Schema View. The default view in which XML Schema documents open can be set in the [File Types tab](#) of the Options dialog. You can switch between views while you edit, using the view that is most useful for the current purpose. XML Schema documents are typically saved with the extension `.xsd` or `.xs`.

Editing in Text View

In Text View an XML Schema is edited as an XML document; the [editing features available for XML documents](#) are also available for XML Schemas. As with all XML documents where the schema is identified and accessible, [Text View entry helpers](#) display the items available for addition at the cursor location point.

Editing in Grid View

In Grid View an XML Schema is edited as an XML document; the [editing features available for XML documents](#) are also available for XML Schemas. When an item in Grid View is selected, [Grid View entry helpers](#) display the items available for addition at the cursor location point.

Editing in Schema View

Schema View is a graphical interface for designing schemas. While you create/edit the schema in Schema View, XMLSpy generates a corresponding text document behind the interface. How to create and edit XML Schema documents in Schema View is described in detail in the section [Editing Views | Schema View](#).

XML Schema features in XMLSpy

Additionally, XMLSpy offers the following very useful features:

- *Convert XML Schema to DTD*: With the [DTD/Schema | Convert DTD/Schema](#) command, XML Schemas can be converted to DTDs.
- *Generate sample XML file from XML Schema*: With the [DTD/Schema | Generate Sample XML File](#) command, an XML document can be generated that is based on the active XML Schema. Sample values can also be specified for elements and attributes in the sample XML.
- [XML signatures](#) for XML Schema (`.xsd`) files in Schema View can be created as external signature files. How to work with signatures is described in the section, [XML Signatures](#).

6.3 Schema Subsets

One or more components of an XML Schema can be created as a separate schema file, known as a schema subset. The advantage of using smaller schema subsets to compose the larger schema (by means of Includes) is that the smaller files are more manageable than the single full schema.

In Schema View, one possible work scenario that describes various aspects of the Schema Subsets feature is as follows:

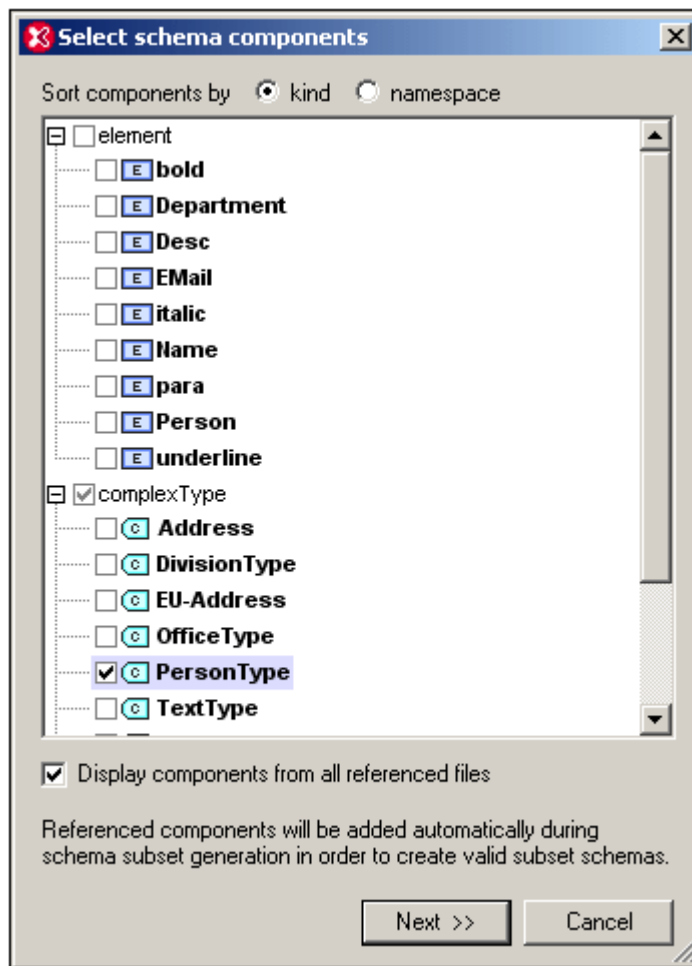
1. Create a schema subset that contains one or more components of the active schema. How to do this is [described below](#),
2. Create additional schema subsets as required.
3. Include the newly created schema subset/s to compose the larger schema. Do this for each schema subset by appending or inserting an Include component in the [Schema Overview window](#), and selecting the newly created schema subset file.
4. Delete any components that were present in the original full schema but are now duplicated because of the included subset/s.

You can also do the reverse in Schema View, that is, flatten the included schema subsets so that: (i) the components contained in the schema subsets are added directly to the main schema, and (ii) the included schema subsets are deleted from the main schema. How to flatten a schema is [described further below](#).

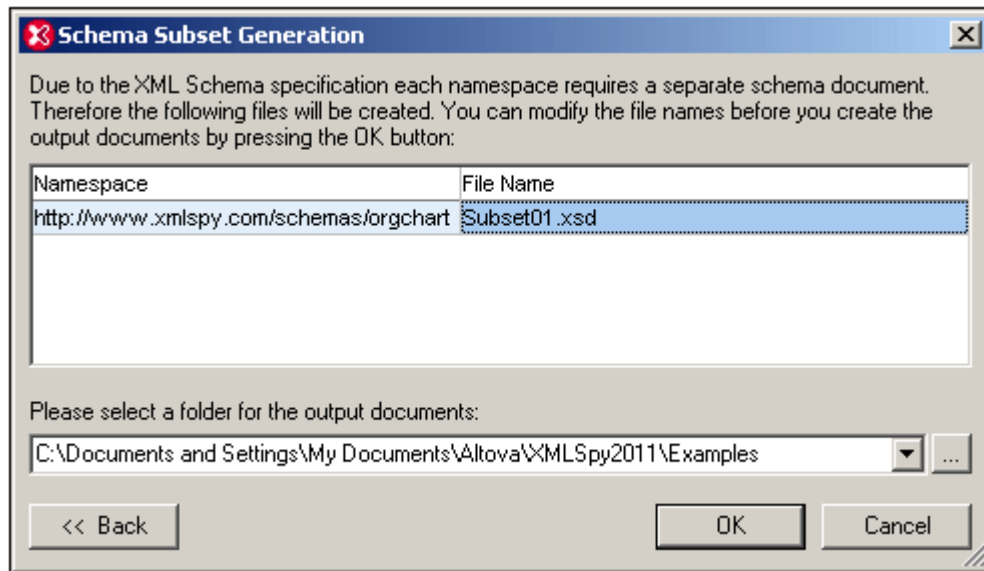
Creating schema subsets

To create a schema subset, do the following:

1. With the required XML Schema active in Schema View, select the command **Schema Design | Create Schema Subset**. This pops up the Select Schema Components dialog (*screenshot below*).
2. In the dialog, check the component or components you wish to create as a single schema subset, then click **Next**. (Note that a check box below the pane enables components from all referenced files to also be listed for selection.)



3. In the Schema Subset Generation dialog that now appears (*screenshot below*), enter the name/s you want the file/s of the schema subset package to have. You must also specify the folder in which the new schema subset files are to be saved. A schema subset package could have multiple files if one or more of the components being created is an imported component in the original schema. A separate schema file is created for each namespace in the schema subset. The filenames displayed in the dialog are, by default, the names of the original files. But since you are not allowed to overwrite the original files, use new filenames if you wish to save the files in the same folder as the original files.

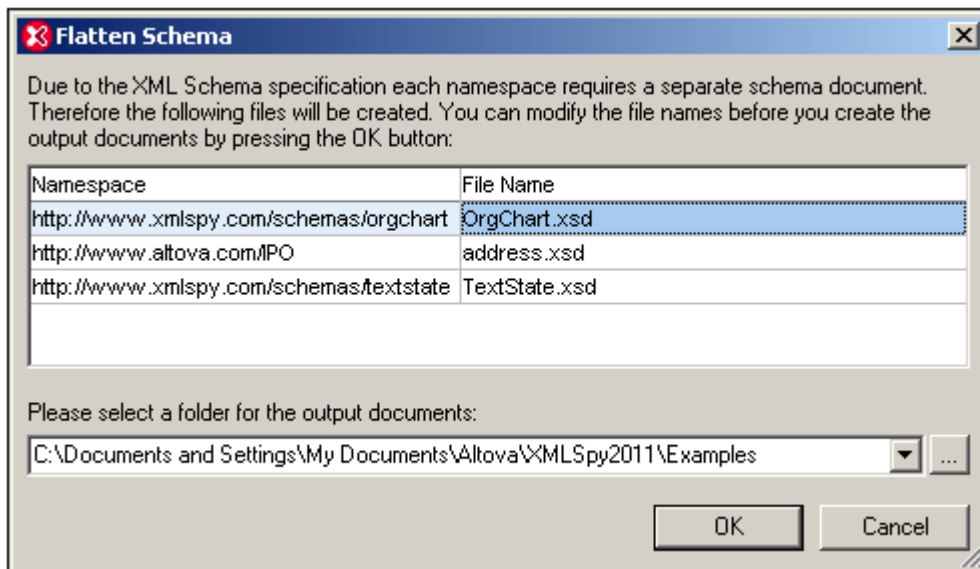


- On clicking **OK**, the schema subset file with the namespace corresponding to that of the active file is opened in Schema View. Any other files in the package are created but not opened in Schema View.

Flattening a schema

Flattening the active schema in Schema View is the process of: (i) adding the components of all included schemas as global components of the active schema, and (ii) deleting the included schemas.

To flatten the active schema, select the command **Schema Design | Flatten Schema**. This pops up the Flatten Schema dialog (*screenshot below*), which contains the names of separate files, one for each namespace that will be in the flattened schema. These default names are the same as the original filenames. But since you are not allowed to overwrite the original files, the filenames must be changed if you wish to save in the same folder as the active file. You can browse for a folder in which the flattened schema and its associated files will be saved.



On clicking **OK**, the flattened schema file will be opened in Schema View.

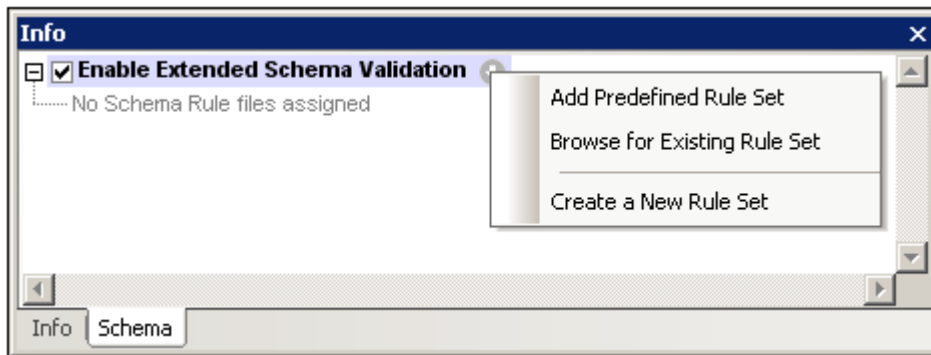
6.4 Schema Rules

A Schema Rule Set is a set of rules one can use to validate an XML Schema. For example, a rule can specify that attribute names, as defined in the XML Schema, begin with a lowercase alphabet, or that a given complex type can only be extended from a specific type.

A set of schema rules is saved in a Rule Set file, which is an XML (.xml) file. XMLSpy contains a [Schema Rules Editor](#) in which you can edit schema rules graphically. In XMLSpy, one or more Rule Set files can then be [assigned to an XML Schema](#). The XML Schema is validated in Schema View against the assigned Rule Sets by selecting the **XML | Validate (F8)** command.

6.4.1 Managing Rule Sets

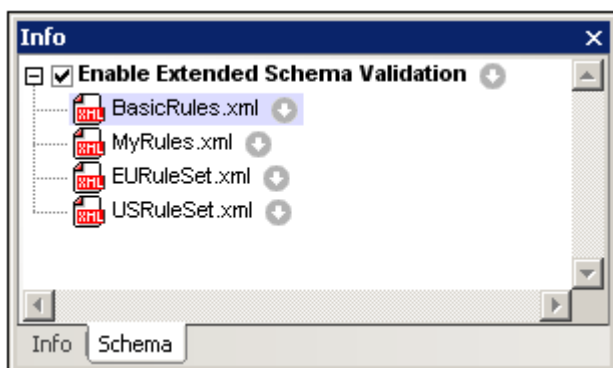
One or more Schema Rule Set files (.xml files) can be assigned to the active XML Schema (.xsd file). This is done via the Schema tab of the Info Window (*screenshot below*).



Adding Rule Sets for extended validation

To add a Schema Rule Set file, click the context menu  button. This pops up a menu (see *screenshot above*) in which you can select how you wish to add Schema Rule Set files to the XML Schema. The following options are available:

- **Add Predefined Rule Set:** You can select from a list of predefined Schema Rule Sets that have been supplied with XMLSpy. These Rule Set files are saved in the `Extended Schema Validation` folder in the XMLSpy application folder. Any Rule Set file added to this folder will be displayed in the Predefined Rule Set dialog and will be available for addition.
- **Browse for Existing Rule Set:** You can browse for a non-predefined Schema Rule Set file.
- **Create a New Rule Set:** Pops up the Schema Rule Editor, in which you can edit the Schema Rules in a Schema Rule Set file. How to work with the Schema Rules Editor is described in the section, [Defining a Rule Set](#). After you save a Schema Rule Set file created via this command, it is added to the listing for the active XML Schema (see *screenshot below*).



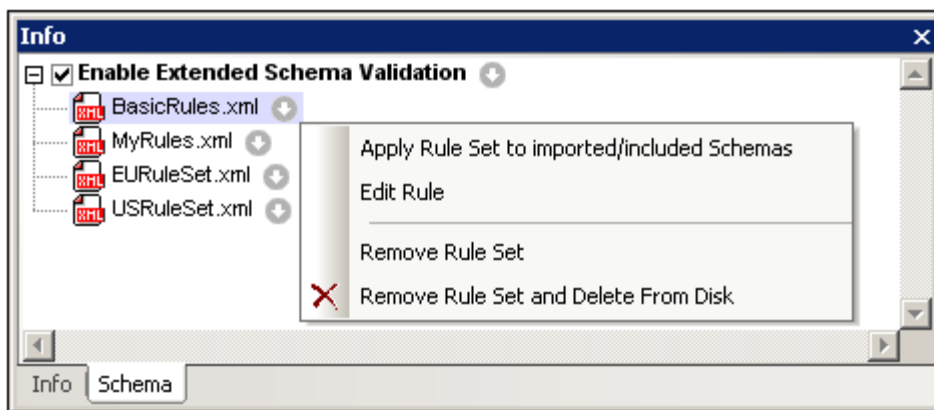
Any number of Schema Rule Sets can be added (see screenshot above). When more than one Schema Rule Set is assigned to an XML Schema, the rules in all the added Schema Rule Sets are used when the XML Schema is validated in Schema View (**XML | Validate**).

Enabling and disabling extended schema validation

Extended schema validation can be enabled or disabled by clicking the Enable Extended Schema Validation check box.

Editing and removing Rule Sets

Individual Rule Sets assigned to an XML Schema can be managed via the context menu that appears on clicking the context menu  button (screenshot below).



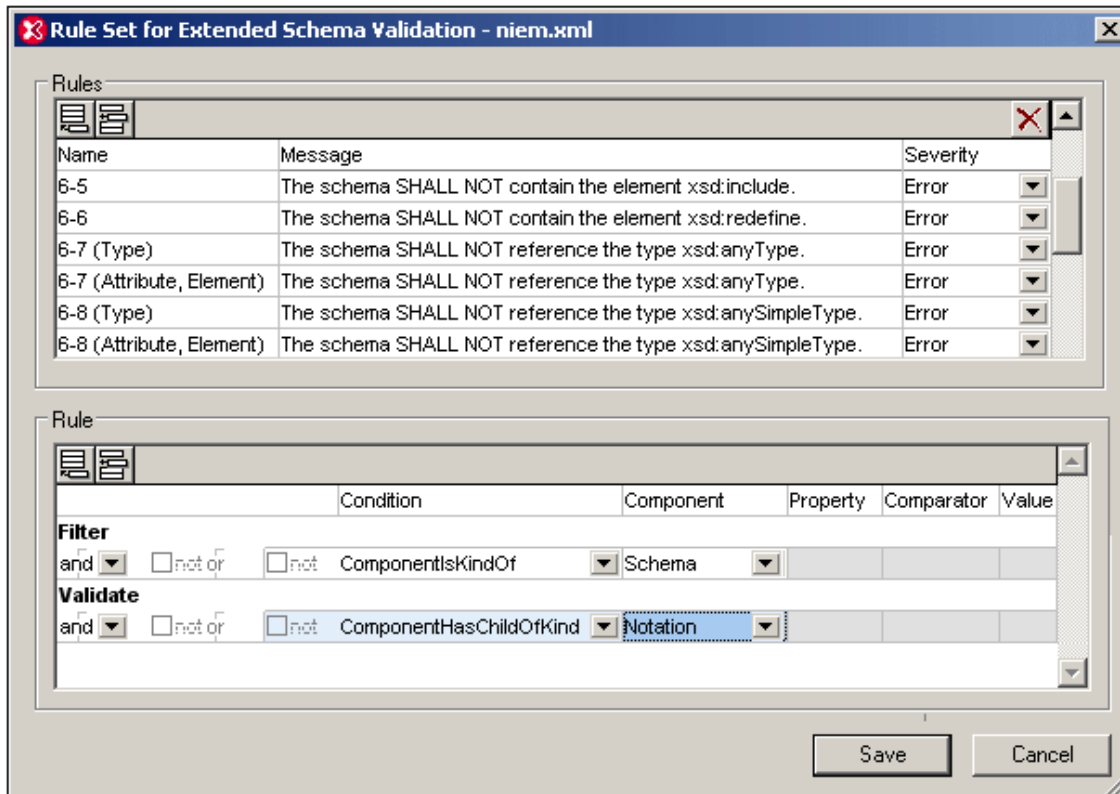
The following options are available:

- *Apply Rule Set to imported and included schemas*: If a Rule Set is applied, rules in it will be used for all schemas that the main schema imports or includes.
- *Edit Rule*: Opens the Schema Rule Set in the Schema Rules Editor.
- *Remove Rule Set*: Removes the Rule Set from the list of added Rule Sets.
- *Remove Rule Set and delete from disk*: This command is enabled for all non-predefined Rule Sets. In addition to removing the Rule Set from the list of added Rule Sets, this command also deletes the Rule Set.

6.4.2 Defining a Rule Set

A Schema Rule Set can be opened for editing in the Schema Rules Editor (*screenshot below*). You can then create, edit, and delete schema rules in that Schema Rule Set file. To open a Rule Set in the Schema Rules Editor, do the following:

1. Select the Rule Set in the list of Rule Sets in the Info window.
2. Clicking the context menu  button of that Rule Set.
3. In the context menu that appears, select Edit Rule.



The Schema Rules Editor dialog has two panes:

- A Rules pane (in the top part of the Editor), in which you can add and delete rules. An empty line for a rule can be appended or inserted by clicking on the respective button (**Append** or **Insert**) in the top left of the pane. A rule can be deleted by selecting it and clicking the **Delete** button in the top right of the pane. Each rule in this pane has a name, a descriptive message text, and a severity level (if the rule is contradicted, validation can be set to return an error or a warning).
- A Rule pane (in the bottom part of the Editor). This pane displays the details of the rule that has been selected in the Rules pane above it, and enables the details of the rule to be edited. For details about defining rules, see the section, [Defining a Rule](#), below.

After the rules in a Rule Set file have been edited, click **Save** to save the rules to the Rule Set File.

Defining a rule

To define or edit a rule, select the rule from the listing in the upper Rules pane. The definition of the rule will be displayed in the Rule pane and can be edited. The screenshot below displays a

rule which can be defined as follows: *If a complex type is an extension of a simple type, then it must have a child kind AttributeGroup.*

		Condition	Component	Property	Comparator	Value
Filter						
☐ not or	☐ not	ComponentIsKindOf	ComplexType			
☐ not or	☐ not	PropertyIsValue		derivedBy	IsEqualTo	extension
☐ not or	☐ not	PropertyResolvesToComponentOfKind	SimpleType	base		
Validate						
☐ not or	☐ not	ComponentHasChildOfKind	Attribute Group			

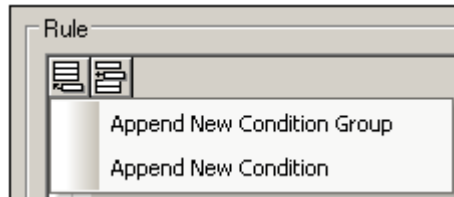
The Validate condition set and Filter condition set:

- Each rule has one set of Validate conditions and one set of Filter conditions (see first column in screenshot above).
- The set of Filter conditions must eventually evaluate to true in order for the Validate condition to be evaluated.
- Each set of conditions (Validate or Filter) consists of one or more Condition Groups, with each Condition Group containing one or more conditions. In the screenshot above, the Validate set contains one Condition Group of one condition, and the Filter set contains three Condition Groups, each having one condition. In the screenshot below, the Filter set contains three Condition Groups: The first contains contains two conditions, the second contains three conditions, and the third contains one condition.

		Condition
Filter		
or	☐ not and	☐ not ComponentIsKindOf
	☐ not and	☐ not ComponentIsKindOf
	☒ not and	☐ not ComponentIsKindOf
	☐ not and	☐ not ComponentIsKindOf
	☐ not and	☐ not ComponentIsKindOf
	☐ not and	☐ not ComponentIsKindOf
Validate		

- Each individual condition can be negated by checking its *Not* check box (located to the left of the condition).
- Within a Condition Group, the logical connectors **and** or **or** indicate, respectively, whether all conditions in the group or one condition in the Condition Group must evaluate to true in order for the entire Condition Group to evaluate to true. In the GUI, these logical operators are the inner of the two columns of logical operators.
- Each Condition Group can be negated by checking its *Not* check box (located to the left of the Condition Group's logical operator).
- The outer logical connector **and** or **or** indicates, respectively, whether all the Condition Groups in the set (Validate or Filter) or one Condition Group must evaluate to true in order for the entire set (Validate or Filter) to evaluate to true.
- Logical connectors can be changed by selecting the appropriate option in the combo box for the outer logical connector (the Condition Group connector). The value of the inner logical connectors (the connectors for conditions within a Condition Group) are all switched to the opposite value as that of the outer logical connector.
- A Condition Group or Condition can be appended or inserted relative to the selected condition. Do this by selecting a condition, then clicking the **Append** or **Insert** button (at

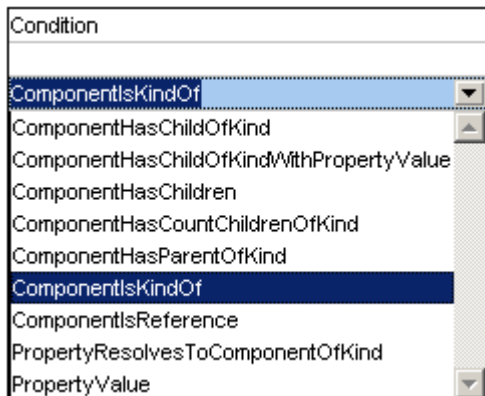
the top left of the pane) and then selecting the required item (Condition Group or Condition) from the menu (see *screenshot below*).



Kinds of condition

A condition can belong to one of three groups (*also see screenshot below*):

- Component kind (in the dropdown list the kinds beginning with *Component*; see *screenshot below*)
- Property kind (*Property Value*)
- A combination of component and property kinds (kinds with *Property* and *Component* in their names)



The kind of a condition is selected from the dropdown list in the *Condition* column of the condition (*screenshot above*). Each of the three groups of conditions is described below.

Conditions of Component kind

For conditions of the Component kind (kinds beginning with *Component*), the component must be specified subsequently in the *Component* column (see *screenshot below*). The component is selected from the dropdown list in the Component field of a sub-condition. Since no other field (Property, Comparator Value) is to be defined for conditions of the Component kind, all other fields are grayed out.

Condition	Component
ComponentIsKindOf	Element
ComponentHasChildOfKind	Attribute Group
	Attribute wildcard
	ComplexType
	Element
	Element wildcard
	Field
	Group
	Import

In the screenshot above, the Filter condition specifies that the rule concerns components of kind *Element*. If the Validate condition then specifies that the component must have a child of kind *Notation*, then the complete rule can be stated as: An *Element* component must have a child of kind *Notation*. If the Validate condition had its **NOT** option checked, then the rule would be stated as: An *Element* component must not have a child of kind *Notation*.

Conditions of Property kind

The condition of the Property kind is *Property Value* (see screenshot below). This kind of condition specifies the nature of a property. It therefore requires entries in the *Property* and *Comparator* columns, and, optionally, an entry in the *Value* column. No entry is required in the *Component* column, which is therefore grayed out. Properties listed in the dropdown lists of the *Property* column include not only XML attributes (such as `default` and `maxOccurs`) but also the logical properties of components (such as `derivedBy`).

	Condition	Component	Property	Comparator	Value
Filter	or ☐ not and ☐ not	ComponentIsKindOf	Model group		
Validate	or ☐ not and ☒ not	Property Value	model	IsEqualTo	all

The screenshot above shows a rule in which the *Model* property has a value equal to *All* and is negated (via the *Not* check box). Taken in conjunction with the filter on the *Model Group* component, this rule simply states that a schema must not contain any `xsd:all` element.

Note: The following points should be noted:

- When using the `IsQNameEqualTo` comparator, the corresponding value must be written in the form: `{ URI } localName`. For example, a value could be: `{ http://www.w3.org/2001/XMLSchema } NOTATION`.
- The `default` property can be present and empty (`<element name default=""/>`) or it can be absent (`<element name/>`).

Conditions that combine Component and Property kinds

Conditions that are a combination of Component and Property kinds are:

- `ComponentHasChildOfKindWithPropertyValue`: Specifies the component kind of a child element and the property's value.
- `PropertyResolvesToComponentOfKind`: A property is specified that resolves to a component kind. The *Comparator* and *Value* columns are empty.

Negating a condition

A condition is negated by checking the *Not* check box to its immediate left (the inner *Not* check boxes). A Condition Group is negated by checking the *Not* check box to the left of the logical connector for conditions in that Condition Group..

6.5 Catalogs in XMLSpy

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

The catalog mechanism in XMLSpy works as outlined below.

RootCatalog.xml

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

```
<?xml 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%` 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 XMLSpy is installed. You should take care not to duplicate mappings in these files, as this could lead to errors.

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

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

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

the folder of a specific schema 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 XMLSpy application folder. The file `CustomCatalog.xml` is located in your `MyDocuments\Altova\XMLSpy` folder. The `catalog.xml` files are each in a specific schema folder, these schema folders being inside the folders: `%AltovaCommonFolder%\Schemas` 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:

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

%LocalAppDataFolder%	Full path to the file system directory that serves as the data repository for local (nonroaming) 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 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 this local file will be used as the DTD. If there is no mapping for the `Public` ID in the catalog, then the URL in the XML document will be used (in the example above:

`http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd`).

The catalog subset supported by XMLSpy

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

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


```
rewritePrefix="Replacement string to locate resource locally"/>
```

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

File extensions and intelligent editing according to a schema

Via catalog files you can also specify that documents with a particular file extension should have XMLSpy's intelligent editing features applied in conformance with the rules in a schema you specify. For example, if you create a custom file extension `.myhtml` for (HTML) files that are to be valid according to the HTML DTD, then you can enable intelligent editing for files with these extensions by adding the following element of text to `CustomCatalog.xml` as a child of the `<catalog>` element.

```
<spy:fileExtHelper ext="myhtml" uri="schemas/xhtml/xhtml1-transitional.dtd"/>
```

This would enable intelligent editing (auto-completion, entry helpers, etc) of `.myhtml` files in XMLSpy according to the XHTML 1.0 Transitional DTD. Refer to the `catalog.xml` file in the `%AltovaCommonFolder%\Schemas\xhtml` folder, which contains similar entries.

XML Schema and catalogs

XML Schema information is built into XMLSpy and the validity of XML Schema documents is checked against this internal information. In an XML Schema document, therefore, no references should be made to any schema for XML Schema.

The `catalog.xml` file in the `%AltovaCommonFolder%\Schemas\schemas` folder contains references to DTDs that implement older XML Schema specifications. You should not validate your XML Schema documents against either of these schemas. The referenced files are included solely to provide XMLSpy with entry helper info for editing purposes should you wish to create documents according to these older recommendations.

More information

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

6.6 Working with SchemaAgent

XMLSpy can be set up to work with Altova's SchemaAgent technology.

SchemaAgent technology

The SchemaAgent technology enables users to build and edit relationships between multiple schemas. It consists of:

- A SchemaAgent Server, which holds and serves information about the relationships among schemas in one or more search path/s (folder/s on the network) that you specify.
- A SchemaAgent client, Altova's SchemaAgent product, which uses schema information from the SchemaAgent server to which it is connected (i) to build relationships between these schemas; and (ii) to manage these schemas (rename, move, delete schemas, etc).

Two types of SchemaAgent server are available:

- Altova SchemaAgent Server, which can be installed on, and accessed from, a network, and
- Altova SchemaAgent, which is the SchemaAgent client product. It includes a lighter server version, called LocalServer, which can only be used on the same machine on which SchemaAgent is installed.

XMLSpy uses SchemaAgent technology to directly edit schemas in Schema View using information about other schemas it gets from a SchemaAgent server. In this setup, XMLSpy is connected to a SchemaAgent server, and, in interaction with SchemaAgent Client, sends requests to SchemaAgent Server. When XMLSpy has been set up to work with SchemaAgent, the Entry Helpers in Schema View not only list components from the schema currently active in Schema View but also list components from other schemas in the search paths of the SchemaAgent server to which it is connected. This provides you with direct access to these components. You can view the content model of a component belonging to another schema in Schema View, and reuse this component with or without modifications. You can also build relationships between schemas, thereby enabling you to modularize and manage complex schemas directly from within XMLSpy.

Installing SchemaAgent and SchemaAgent Server

For details about installing SchemaAgent and SchemaAgent Server and configuring search paths on servers, see the SchemaAgent user manual.

Setting up XMLSpy as a SchemaAgent client

In order for XMLSpy to work as a SchemaAgent client, you must do the following:

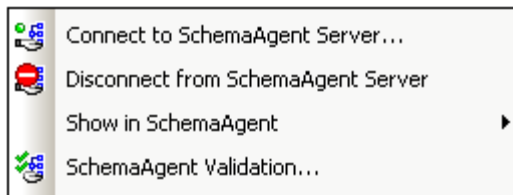
- Download SchemaAgent from the [Altova website](#). You can now use SchemaAgent's LocalServer to serve schemas. For information about configuring search paths on LocalServer, see the SchemaAgent user manual.
Please note: SchemaAgent requires a valid license, which must be purchased after the free trial period runs out. Also note that Altova MissionKit product packages each includes the SchemaAgent product and a license key for it. (The SchemaAgent Server application, however, is not included in Altova MissionKit packages.)
- Additionally, you might want to download and install the network-based SchemaAgent Server from the [Altova website](#).
- Define the search path(s) for SchemaAgent server (also known as configuring SchemaAgent Server). A detailed description of how to do this is given in the SchemaAgent user manual. (A search path is a path to the folder containing the XML

- schemas that will be mapped for their relationships with each other.)
- Start a connection from within XMLSpy to a SchemaAgent server.

Important: All SchemaAgent and SchemaAgent-related products from Altova (including XMLSpy) starting with Version 2005 release 3 are **not compatible** with previous versions of SchemaAgent or SchemaAgent-related products.

SchemaAgent commands in XMLSpy

The SchemaAgent functionality in XMLSpy is available only in Schema View and is accessed via menu commands in the Schema Design menu (see *screenshot*) and by using the Entry Helpers in Schema View.



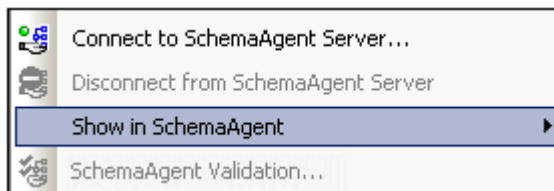
The menu commands provide general administrative functionality. The Entry Helpers (and standard GUI mechanisms, such as drag-and-drop) are used to actually edit schemas.

This section describes how to use the SchemaAgent functionality available in Schema View.

6.6.1 Connecting to SchemaAgent Server

Please note: SchemaAgent Client must be installed in order for you to be able to make a connection.

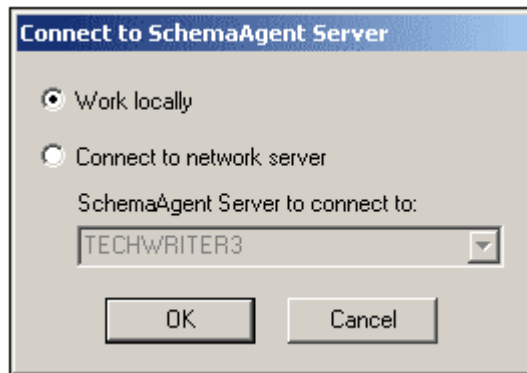
Before you connect to SchemaAgent Server, only the **Connect to SchemaAgent Server** command is enabled in the **Schema Design** menu; other SchemaAgent commands in the **Schema Design** menu are disabled (see *screenshot*). The other menu items become enabled once a connection to a SchemaAgent Server has been successfully made.



Connection steps

To connect to a SchemaAgent server:

1. Click the Connect to SchemaAgent server toolbar icon  (**Schema Design | Connect to SchemaAgent Server**). The Connect to SchemaAgent Server dialog (see *screenshot below*) opens:



2. You can use either the local server (the SchemaAgent server that is packaged with Altova SchemaAgent) or a network server (the Altova SchemaAgent Server product, which is available free of charge). If you select Work Locally, the local server of SchemaAgent will be started when you click **OK** and a connection to it will be established. If you select Connect to Network Server, the selected SchemaAgent Server must be running in order for a connection to be made.

Note on servers running with Windows XP SP2

If the SchemaAgent Server name is listed in the Connect to SchemaAgent Server dialog but you cannot connect to it, it is possible that your server is not taking part in the name resolution process of your network. Name resolution is blocked by the default settings of the Windows XP SP2 Firewall.

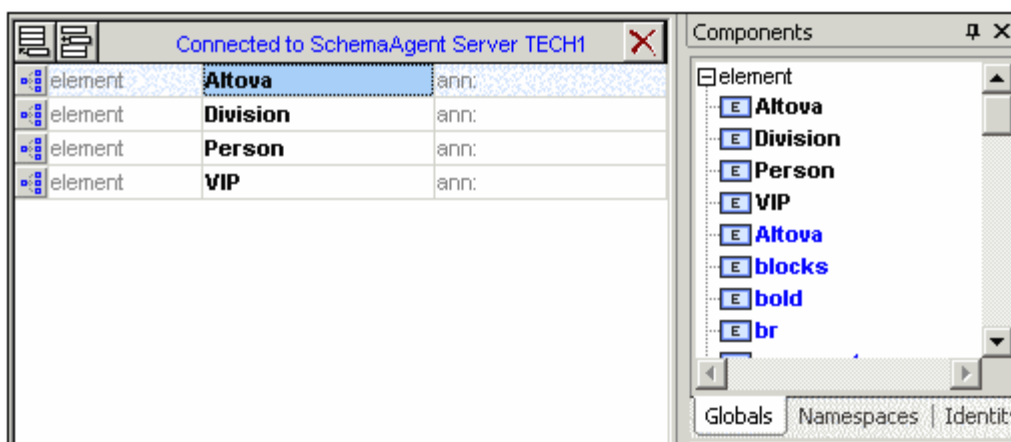
To connect to such a server, do one of the following:

- Change the server settings to enable the name resolution process, or
- Enter the IP address of the server in the Edit field of the Connect Dialog box.

This need be done only once as SchemaAgent Client stores the connection string of the last successful connection.

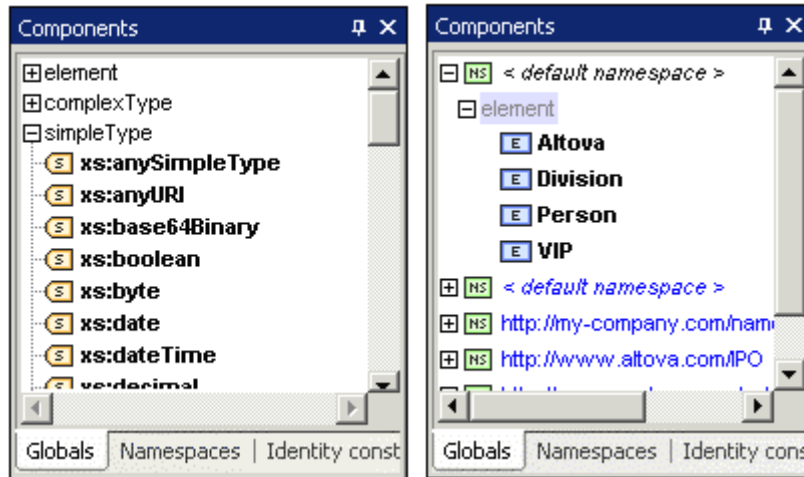
Schema View after connecting to SchemaAgent server

After a connection to a SchemaAgent server is established, Schema View will look something like this:



Please note:

- At the top of the Globals view the text "Connected to SchemaAgent Server" appears, specifying the server to which the connection has been made.
- You now have full access to all schemas and schema constructs available in the server search path. SchemaAgent schema constructs such as global elements, complexTypes, and simpleTypes are visible in **bold blue text**, below the constructs of the active schema (**bold black text**).

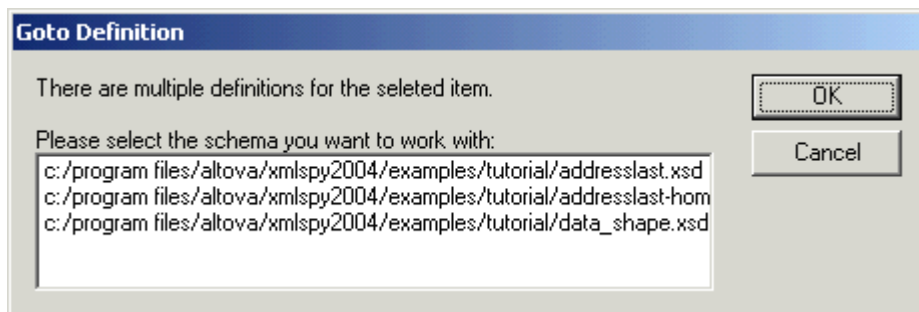


Schema constructs can be viewed by Type (Globals), by Namespace, or by Identity Constraints in the respective tabs in the Schema View.

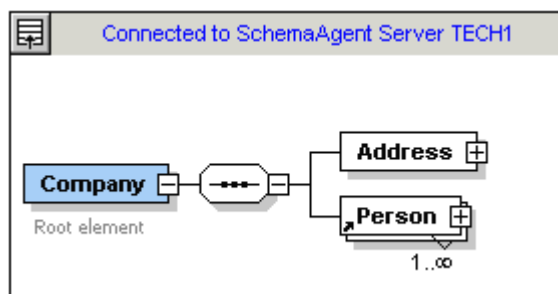
6.6.2 Opening Schemas Found in the Search Path

This example demonstrates how to open a schema found in a search path defined in SchemaAgent Server. It uses the `DB2schema.xsd` file available in the `..\Tutorial` folder as the active schema. The `Global` tab of the Components entry helper is active.

1. Scroll down to the blue `Company` entry in the Components entry helper, and double-click it. The Goto Definition dialog box is opened.



2. Click the `Addresslast.xsd` entry, and click **OK** to confirm. This opens the `addresslast.xsd` schema and displays the content model of the `Company` element.



Please note: Double-clicking a SchemaAgent schema construct, such as Element, complexType, or simpleType, opens the associated schema (as well as all other included schemas) in XMLSpy.

6.6.3 Using IIRs

XML schema provides Import, Include, and Redefine (IIR) statements to help modularize schemas. Each method has different namespace requirements. These requirements are automatically checked by SchemaAgent Client and XMLSpy when you try to create IIRs.

Imports, Includes, and Redefines (IIRs)

Schema constructs can be "inserted" by different methods:

- Global elements can be dragged directly from the Components Entry Helper into the content model of a schema component (in Schema View).
- Components, such as complexTypes and simpleTypes, can be selected from the list box that automatically opens when defining new elements/attributes, etc.
- Components, such as complexTypes, can be selected from the Details Entry Helper when creating/updating these type of constructs.

Incorporating schema components

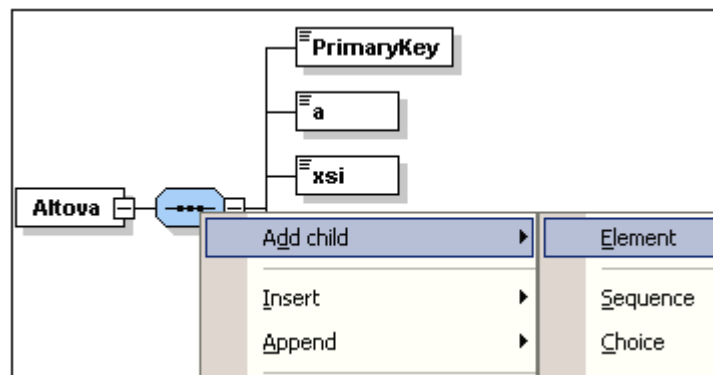
This example uses the `DB2Schema.xsd` file available in the `.. \Tutorial` folder as the active schema; the `Global` tab of the Components Entry Helper is active.

To use schema constructs from SchemaAgent Server schemas:

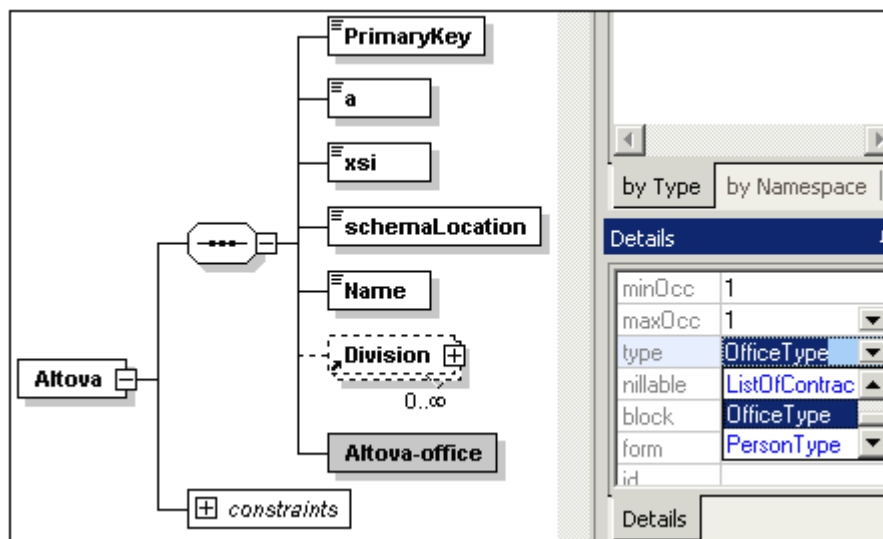
1. Make sure you are connected to a SchemaAgent server (see [Connecting to SchemaAgent server](#)).
2. Open and rename the `DB2Schema.xsd` file for this example, for example to `Altova-office`.

Connected to SchemaAgent Server TECH1		
element	Altova	ann:
element	Division	ann:
element	Person	ann:
element	VIP	ann:

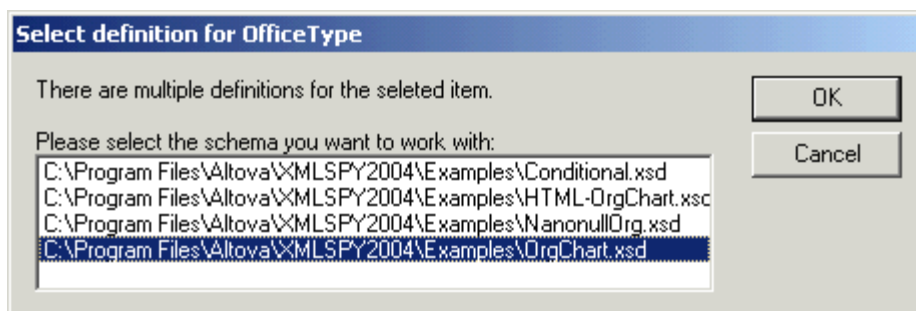
3. Click the  icon of the `Altova` element in the Schema Overview to see its content model.
4. Right-click the `Altova` sequence compositor and select the menu option **Add Child | Element**. Note that a list box containing all global elements within the server path opens automatically at this point. Selecting one would incorporate that element.



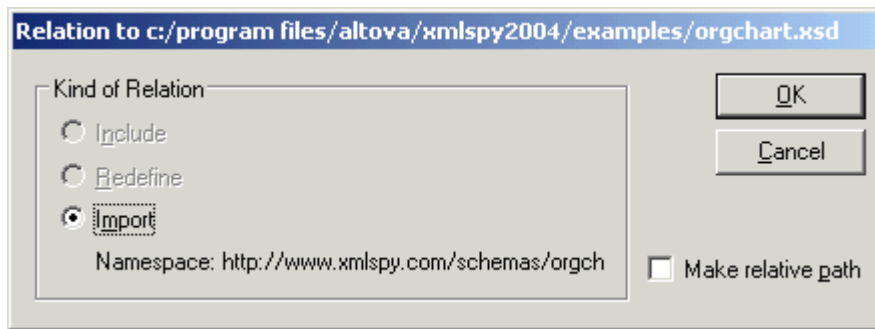
5. Enter `Altova-office` as the name for this new element and press **Enter**.
6. Using the Details Entry Helper, click the `type` combo box and select the entry `OfficeType`.



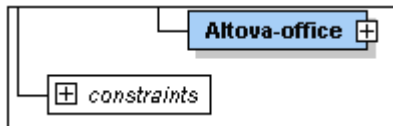
This opens the Select Definition For OfficeType dialog box.



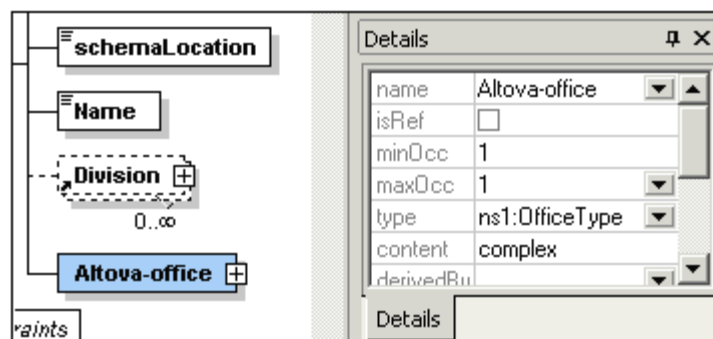
7. Select `Orgchart.xsd` and click **OK** to confirm.



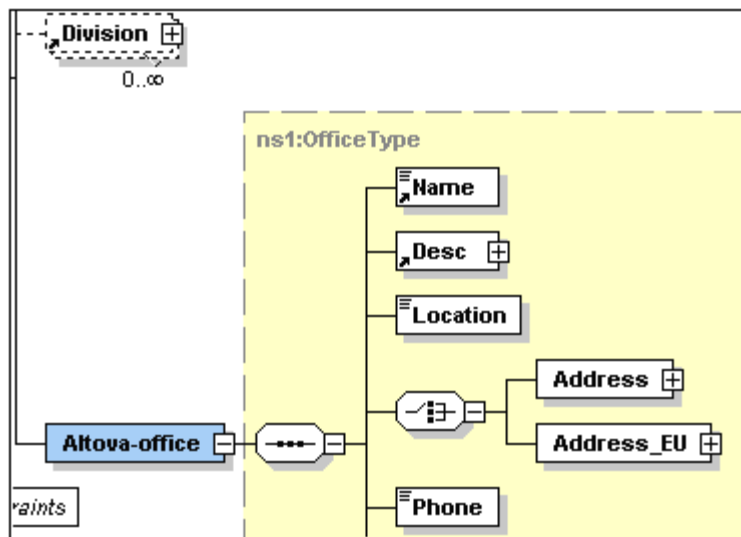
8. Click **OK**. The Import command was automatically selected for you. An expand icon appears in the `Altova-office` element.



Please note: The `type` entry in the Details entry helper has changed; it is now displayed as `ns1:OfficeType` due to the fact that the `Orgchart.xsd` schema file has been imported and the target namespaces must be different in both schemas. An Import command has also been added to the schema.



9. Click the Expand button to see the `OfficeType` content model.

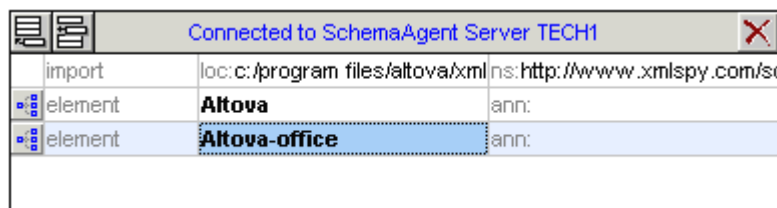


10. Press **F8** to validate the schema. The "Schema is valid" message should appear at this

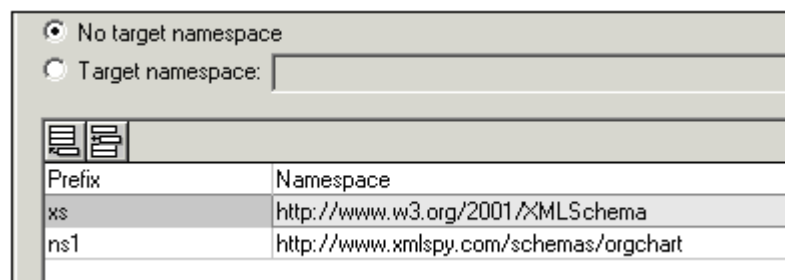
stage.

Cleaning up the schema:

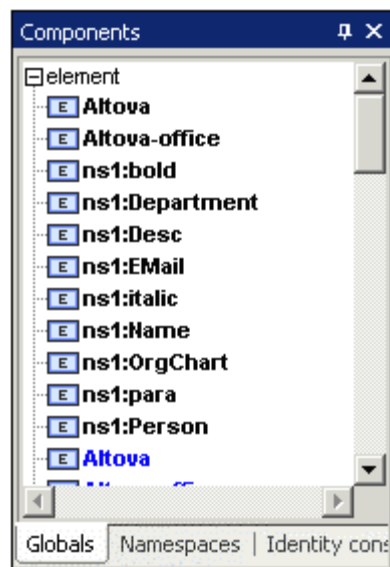
1. Delete the `Division` element in the content model.
2. Click the Return to globals icon  to switch to the Schema Overview.
3. Delete the following global elements: `Division`, `Person` and `VIP`.



4. Select the menu option **Schema Design | Schema settings** to see how the namespace settings have changed.



The `ns1` prefix has been automatically added to the `www.xmlspy.com/schemas/orgchart` namespace. The Components (see screenshot) and Details Entry Helpers displays all imported constructs with the `ns1:` namespace prefix.



Please note:

- Changes made to schemas under SchemaAgent Server control using XMLSpy

automatically update other schemas in the SchemaAgent Server path that referenced the changed schema.

- It is possible to see duplicates of constructs element, simpleTypes etc. in entry helpers (in black and blue), if the schema you are working on is also in the SchemaAgent Server path.

6.6.4 Viewing Schemas in SchemaAgent

To work with the active schema and its related schemas in SchemaAgent, select the menu option **Schema Design | Show in SchemaAgent | *schema or related schemas*** (see *screenshot*).

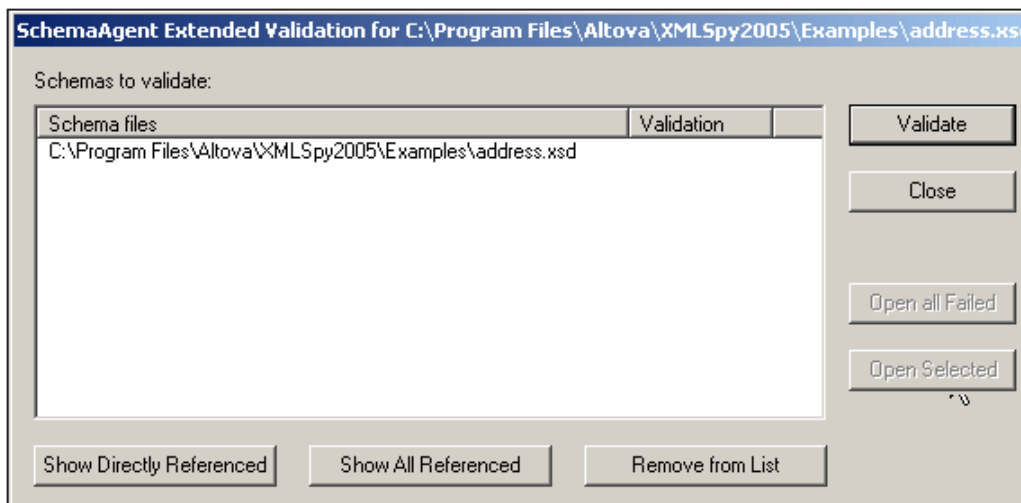
You have the option of opening only the active schema in SchemaAgent (**File only** command), or the active schema together with either (i) all directly referenced schemas, or (ii) all directly referencing schemas, or (iii) all directly related schemas.

6.6.5 SchemaAgent Validation

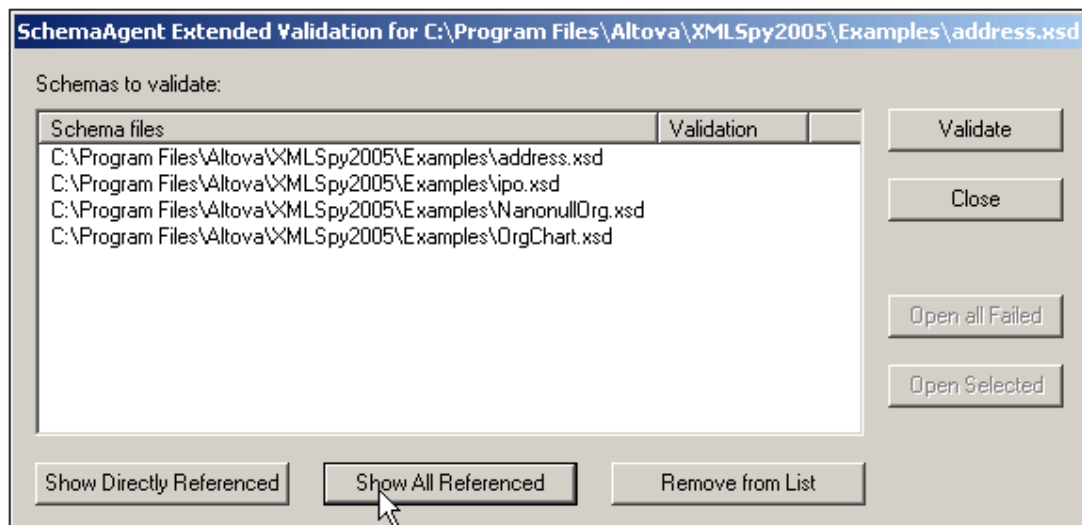
XMLSpy, in conjunction with SchemaAgent, allows you to validate not only the currently active schema but also schemas related to the currently active schema. We call this extended validation. There are two types of related schemas that SchemaAgent distinguishes for extended validation: (i) directly referenced schemas, and (ii) referenced schemas.

How to carry out extended validation is demonstrated below by means of an example. This example assumes that the schema file `address.xsd` is the active schema in Schema View of XMLSpy. You would then do the following:

1. Click the Extended Validation icon  in the toolbar or the menu item **Schema Design | Extended Validation**. This opens the Extended Validation dialog box, in which you can choose whether to validate the active schema only or one or more related schemas as well.

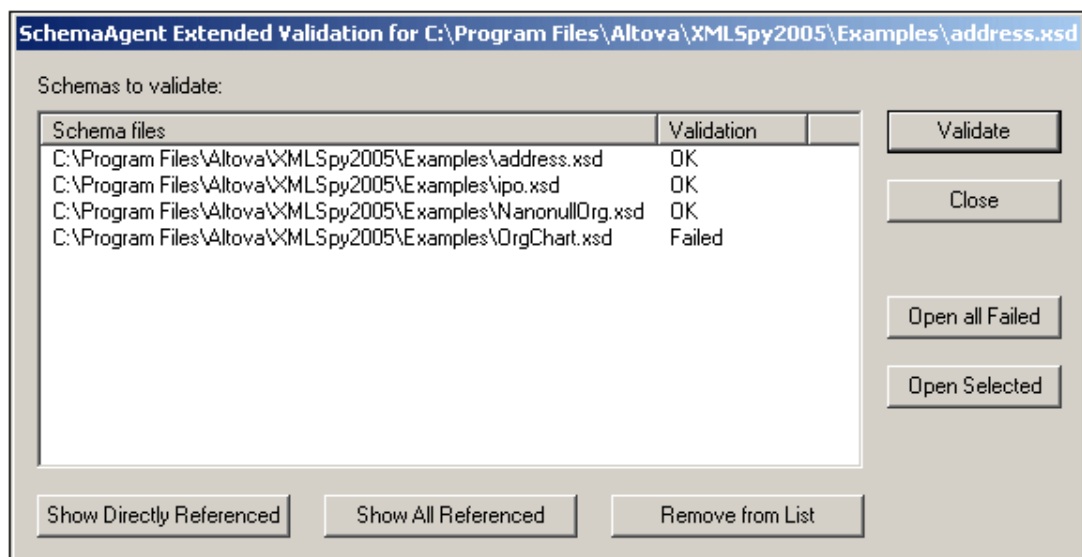


2. To insert schemas into the list, click the **Show Directly Referenced** or **Show All Referenced** button as required. In this example, we have clicked the **Show All Referenced** button, and this inserts all referenced files into the list.



At this point, you can remove a schema from the list (Remove from List).

3. Click the **Validate** button to validate all the schemas in the list box.



The Validate column displays whether the validation was successful or whether it failed.

You can now open the schemas that failed validation or a set of selected schemas in XMLSpy.

6.7 Find in Schemas

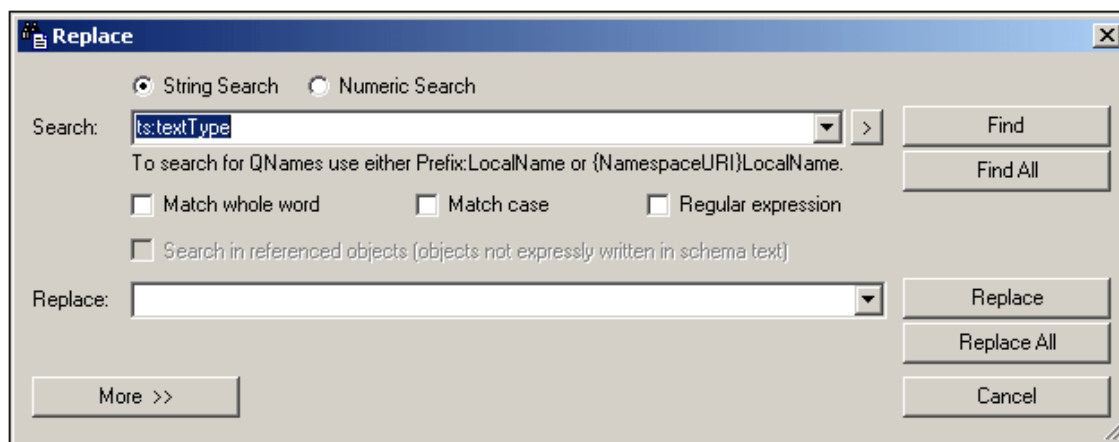
In Schema View, XML Schemas can be searched intelligently using XMLSpy's Find & Replace in Schema View feature.

The Find and Replace in Schema View feature is enabled when a schema is active in Schema View. It is accessed in one of two ways:

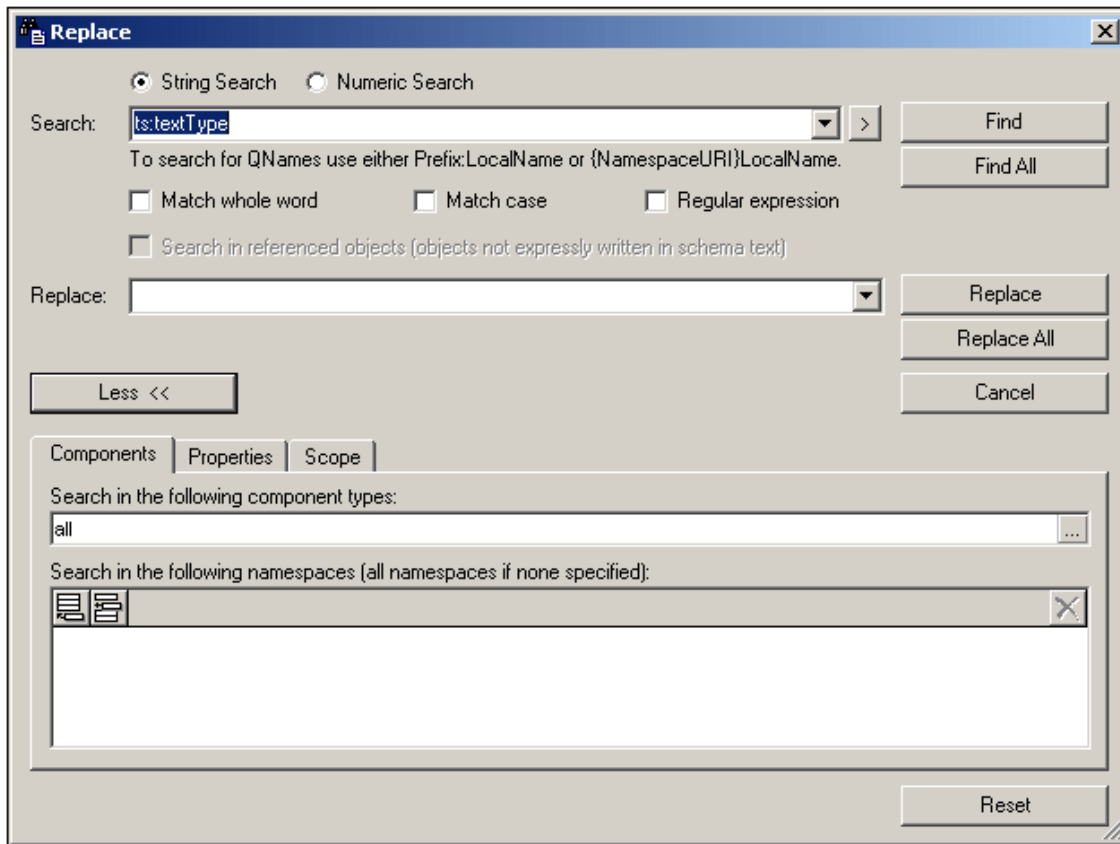
- Via the **Edit | Find** and **Edit | Replace** menu commands.
- Via the **Find** and **Replace** buttons in the Find in Schemas window.

Clicking a command or a button pops up the Find or the Replace dialog, according to which command/button was clicked. The Replace dialog (*screenshots below*) is different from the Find dialog in that it has a text entry field for the Replace term.

The standard Replace dialog looks like this:



Clicking the **More** button expands the dialog to show additional search criteria (*screenshot below*).



Usage is as follows:

- [Enter the search and replace terms](#) in the Search and Replace text fields
- [Specify the schema components to be searched](#) in the Components tab
- [Specify the properties of the components to be searched](#); this helps to narrow the search
- [Set the scope of the search](#) to the current document or project, or specify a folder to search
- [Execute the command](#)
- [Use the Find in Schemas window](#) to navigate to a component quickly

The **Reset** button at the bottom of the dialog resets the original settings, which are as follows:

- No search term, no replace term
- Components: `all`
- Namespaces: none specified
- Property restrictions: `anywhere`
- Additional property restrictions: none
- Scope: current file

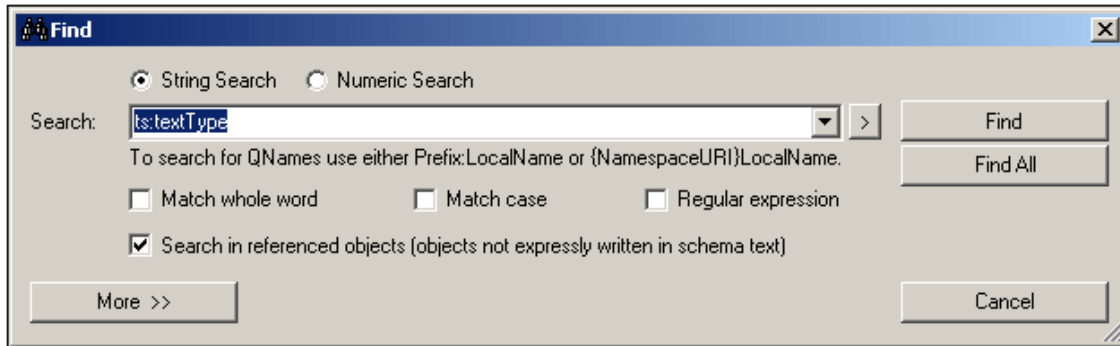
Note: Regular expressions are not supported in the Replace field.

6.7.1 Search Term

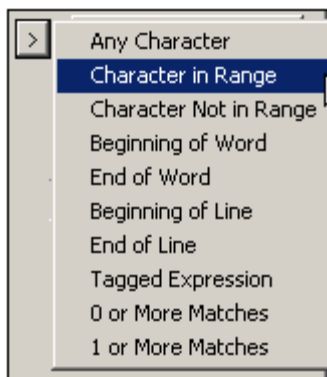
The search term can be entered as a string (select the **String** radio button) or as a number (**Numeric** radio button).

String search

In a string search (*screenshot below*), the entry can be: (i) text; (ii) a QName; or (iii) a regular expression. For QName searches, the namespace is determined on the basis of either the prefix used in the document or by the namespace URI, either of which must be entered. In the screenshot below, the `ts:` prefix is the prefix used in the document to identify a certain namespace.



To search using a regular expression, check the Regular Expression check box and then enter the regular expression. Entry helpers for regular expressions are available in a menu that is activated by clicking the right-pointing arrowhead at the right of the Search entry field (*screenshot below*).

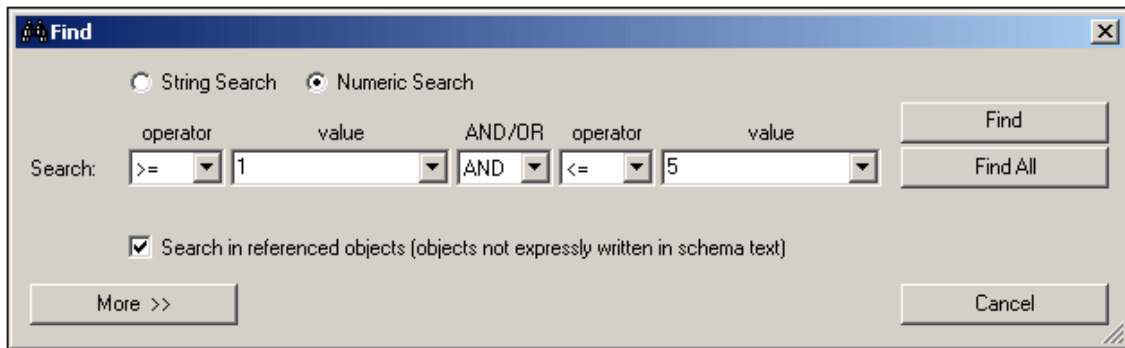


You can also select whether a search term must match a whole word in the document and/or whether the casing in the document must match. Use the check boxes below the text entry field to specify these options.

If you wish to search in referenced objects (such as a complexType definition or a global element), then check the Search In Referenced Objects check box. This option is available only in the Find dialog; it is disabled in the Replace dialog.

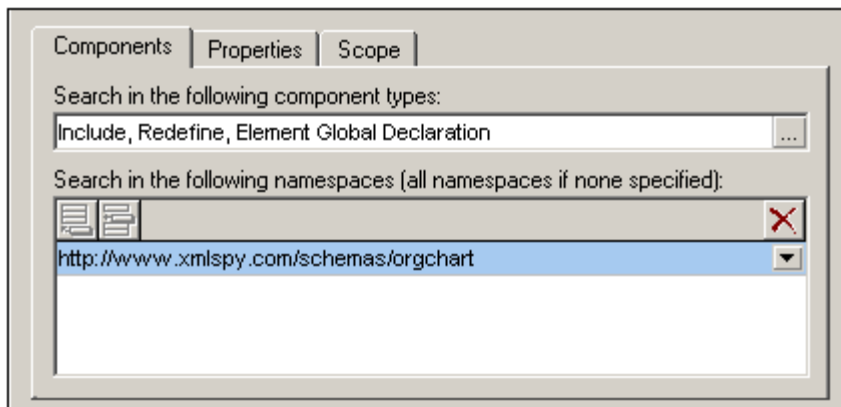
Numeric search

When the Numeric Search radio button is selected, the search term can be a single operator-and-number search parameter, or a set of two such operator-and-number search parameters joined by the logical connector `AND` or `OR`. In the screenshot below, there are two search term parameters which create a search term for all integers between, and including, 1 and 5.



6.7.2 Components

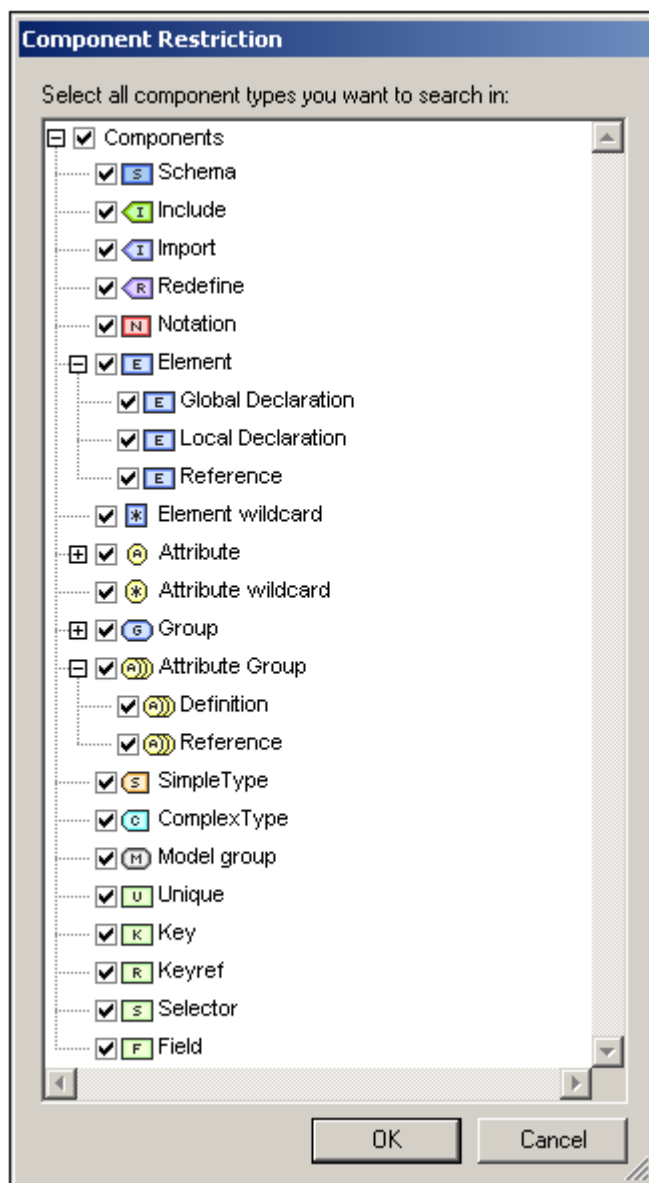
The search can be restricted to one or more component types and to one or more target namespaces. These options are available in the Components tab. Expand the Find or Replace dialog by clicking the **More** button. This will bring up the tabs for refining the search, one of which is the Components tab (*screenshot below*).



The Components tab consists of two parts: (i) for selecting the component types to be searched, and (ii) for selecting the target namespaces to be searched.

Component selection

You can enter the component types to be searched by clicking the **Add** icon  located to the right of the text field (see *screenshot above*). This pops up the Component Restriction dialog (*screenshot below*), in which you can select the components to be searched by checking them. Checking the `Components` item at the top of the list selects all components (text entry: `all`). Unchecking it de-selects all components (text entry: `none`)—including individually selected components. Individual components, therefore, can be selected only when the `Components` item is unchecked. The selected components are entered in the text field as a comma-separated list (see *screenshot above*).



Note: Each time the Components tab or the Find/Replace dialog is opened, the previous component selection is retained.

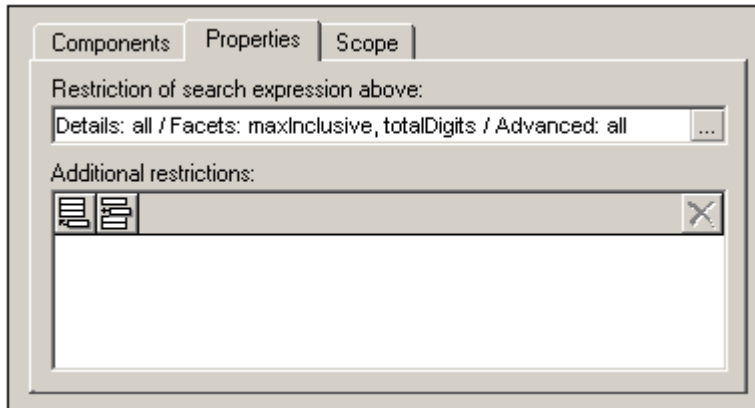
Namespace selection

To select one or more target namespaces to be searched, click the **Add** or **Insert** icons and enter the required namespace/s. If no target namespace is specified, then all target namespaces are searched. To delete a target namespace that has been entered in this pane, select the target namespace and click the **Delete** icon.

6.7.3 Properties

The search can be restricted to one or more component properties (details and facets) by using options in the Properties tab, as well as to match the contents of properties. Expand the Find or Replace dialog by clicking the **More** button, and then select the Properties tab (*screenshot*

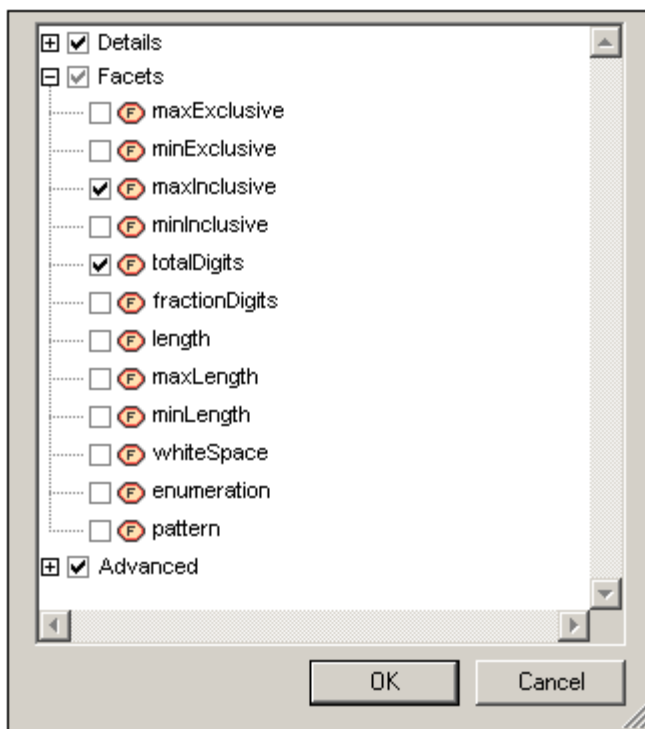
below).



The Properties tab consists of two parts: (i) for restricting the main search term (entered in the Find text box); and (ii) for adding additional content restrictions (which have their own match term); see the section [Additional Restrictions](#) below.

Properties selection

You can enter the property types to be searched by clicking the **Add** icon , which is to the right of the text field (see *screenshot above*). This pops up the Property Restriction dialog (see *screenshot below*), in which you can select the properties to be searched by checking them. The properties are organized in three groups: (i) Details; (ii) Facets; (iii) Advanced (such as the DerivedFrom property). Checking Details, Facets, or Advanced selects all properties in that group. Unchecking a group de-selects all properties in that group, including individually selected properties. Individual properties, therefore, can be selected only when the group item is unchecked. The selected properties are entered in the text field (see *screenshot above*).

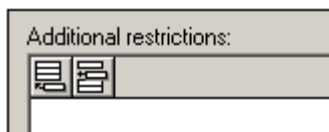


Note: Each time the Properties tab or the Find/Replace dialog is opened, the previous properties selection is retained.

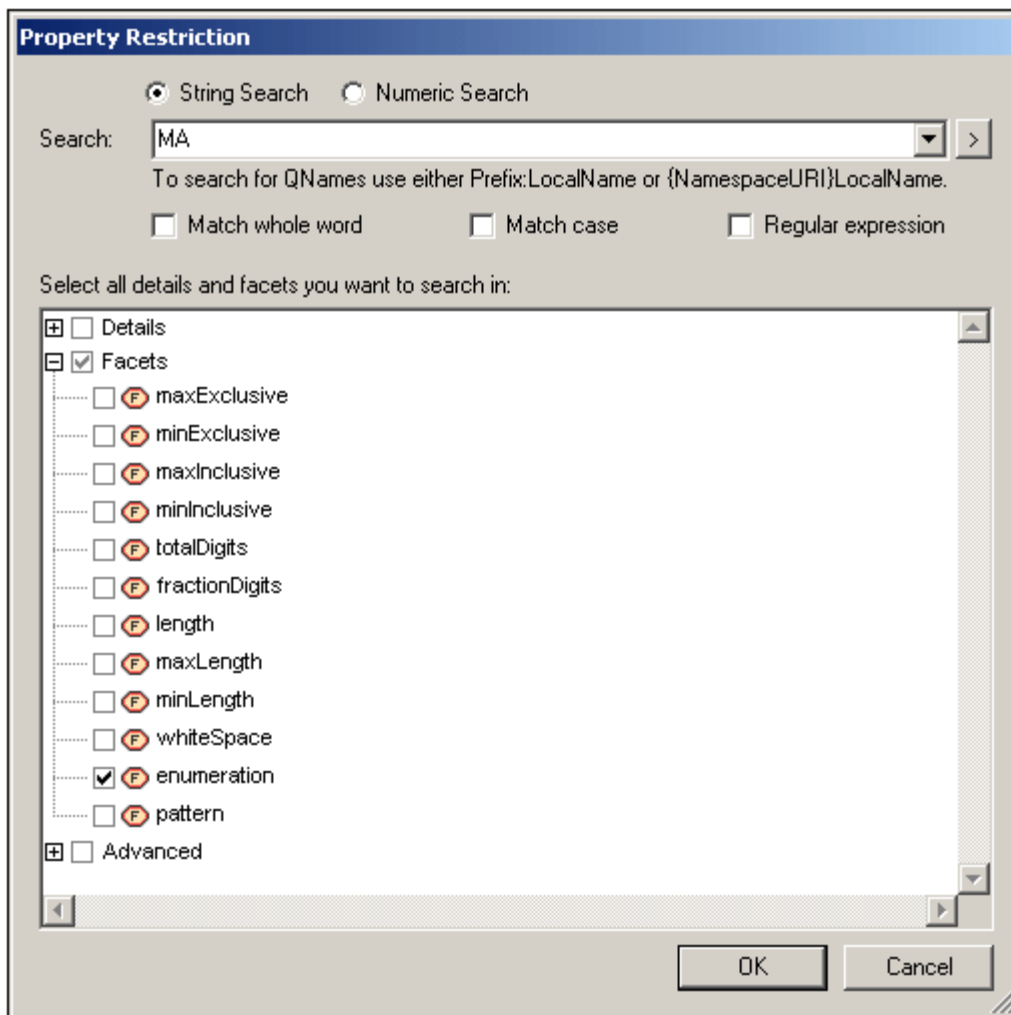
Additional restrictions

An additional restriction enables you to specify the value of the property to search for. For example, if you are looking for an element called `state` which has an enumeration `MA` (for the US state of Massachusetts), you could specify the value `MA` of the property `enumeration` with the Addition Restrictions option. You would do this as follows:

1. In the Additional Restrictions pane, click the **Add** or **Insert** icon (*screenshot below*).

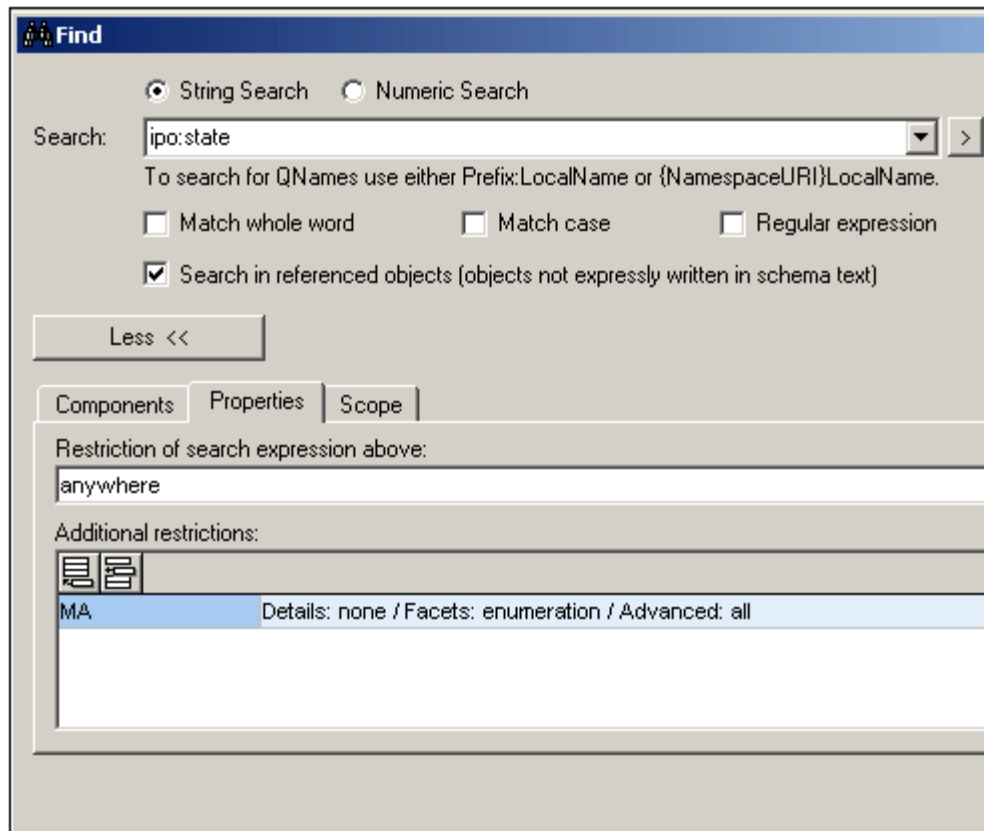


2. This adds a row to the pane and pops up the Property Restriction dialog. Deselect all properties and select only the `enumeration` property (*screenshot below*).



3. In the text field at the top of the dialog, enter the enumeration value to be searched for, in this case, `MA` (*see screenshot above*).

- Click **OK**. The additional restriction is entered in the newly created row in the Additional Restrictions pane (*screenshot below*).



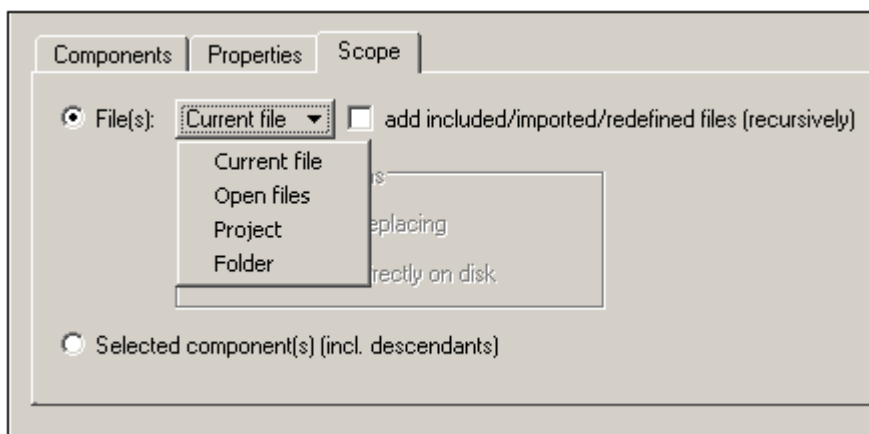
In the screenshot above, notice that the search term is `ipo:state`. In the Properties tab, the `anywhere` specifies that all properties will be searched, but the additional restriction specifies that the search should be restricted to enumerations having a value of `MA`.

Multiple additional restrictions can be added to further narrow the search. To delete an additional restriction, select the additional restriction and click the **Delete** icon.

Note: Each time the Properties tab or the Find/Replace dialog is opened, the previous additional restriction/s are retained.

6.7.4 Scope

The scope of the search can be set in the Scope tab (*screenshot below*). You can select either file/s or the currently selected schema component in Schema View.



If the File/s option is selected, you can further specify one from among the following options:

- *Current file*: An additional option to search included, imported and redefined files is also available.
- *Open files*: All XML Schema (XSD) files that are open in XMLSpy. Only the Find All and Replace All commands are enabled; single-step searching is not available.
- *Project*: The currently active project is selected, with the option to skip external folders. Only the Find All and Replace All commands are enabled; single-step searching is not available. If the default view for the .xsd file extension (**Tools | Options | File Types | Default View**) is not Schema View, then the .xsd files are not searched.
- *Folder*: You can browse for the required folder; an option to search sub-folders is also available. Only the Find All and Replace All commands are enabled; single-step searching is not available. If the default view for the .xsd file extension (**Tools | Options | File Types | Default View**) is not Schema View, then the .xsd files are not searched.
- *Included, imported, and redefined files* can be included in the scope by checking the option for adding them to the scope.

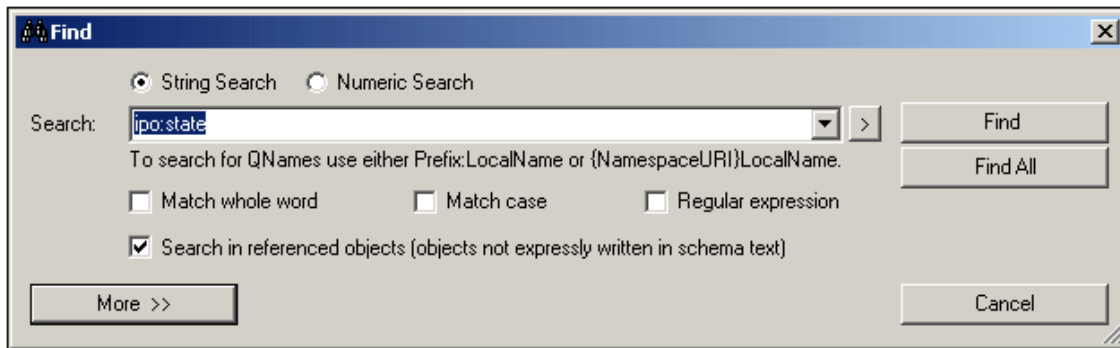
In the Replace dialog, you can choose whether to copy the replacement to the file on disk or whether to open the file in XMLSpy. Do this by selecting the appropriate button in the dialog.

6.7.5 Find and Replace Commands

The **Find** command behaves differently in the Find and Replace dialogs. The behavior of the **Find** command in both dialogs and of the **Replace** command is described below.

Find dialog

After you have entered the search term and, optionally, other criteria to refine the search, you can click either the **Find (F3)** or **Find All** command (*screenshot below*).



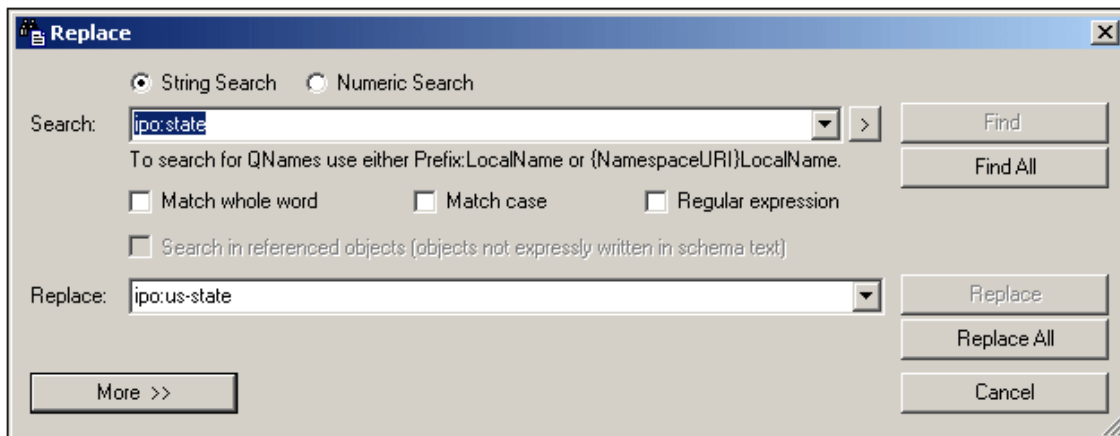
Clicking the **Find (Ctrl+F)** command in the dialog closes the Find dialog and finds the next occurrence of the search term within the specified scope and refinement criteria. The next occurrence is found relative to the currently selected component in Schema View. If the search reaches the end of the scope, it will not start automatically from the beginning of the scope. Therefore, you should make sure that the currently selected component in Schema View before starting the search is located before the document part you wish to search.

The result of the **Find** is highlighted in Schema View and the result is also reported in the Find In Schemas window. In the Find In Schemas window, you can click a result to highlight that item in Schema View.

Clicking the **Find All** command closes the Find dialog and lists all the search results in the Find In Schemas window.

Replace dialog

In the Replace dialog (*screenshot below*), clicking the **Find** command finds the next occurrence of the search term relative to the current selection in Schema View. You can then click **Replace** to replace this occurrence.



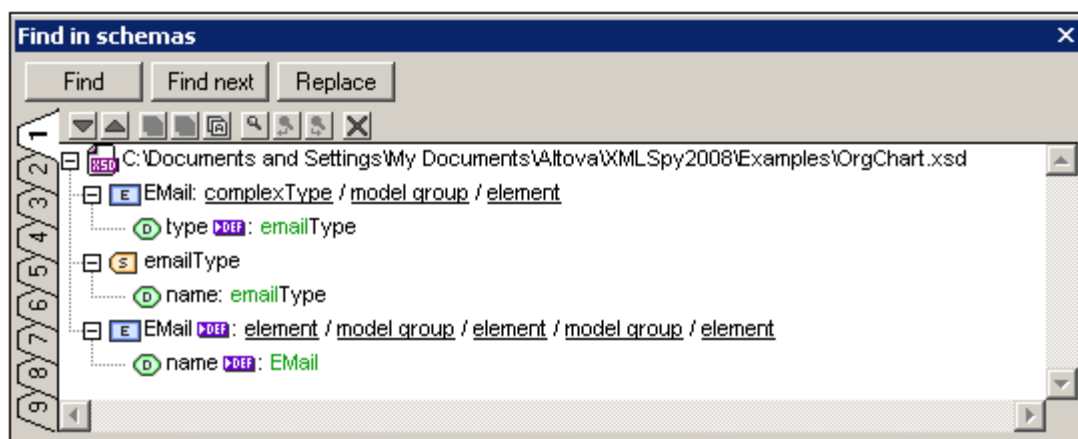
The **Find All** command closes the Replace dialog and lists all the search results in the Find In Schemas window.

The **Replace All** command replaces all occurrences of the found term, closes the Replace dialog, and lists the found terms in the Find In Schemas dialog.

Note: Regular expressions are not supported in the Replace field.

6.7.6 Results and Information

Each time a **Find**, **Find All**, **Replace**, or **Replace All** command is executed the results of the command execution are displayed in the Find In Schemas window (*screenshot below*). The term that was searched for is displayed in green; (in the screenshot below, it can be seen that `email` was the search term, with no case restriction specified). Notice that the location of the schema file is also given.



The **Find All** and **Replace All** commands list all the found occurrences in the document.

Note: The **Find** and **Replace** buttons at the top of this window bring up the Find dialog and the Replace dialog, respectively. The **Find** button can be used to find the next occurrence of the search term.

Features of the Find In Schemas window

Results are displayed in nine separate tabs (numbered 1 to 9). So you can keep the results of one search in one tab, do a new search, and compare results. Clicking on a result in the Find In Schemas window pops up and highlights the relevant component in the Main Window of Schema View. In this way you can search and navigate quickly to the desired component.

The following Find In Schema toolbar commands are available:

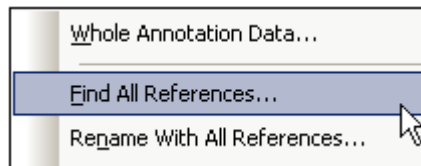
- The **Next** and **Previous** icons select, respectively, the next and previous messages to the currently selected message.
- The **Copy Messages** commands copy, respectively, the selected message, the selected message and its children messages, and all messages, to the clipboard.
- The **Find** commands find text in the Find In Schemas window.
- The **Clear** command deletes all messages in the currently active tab.

6.7.7 Finding and Renaming Globals

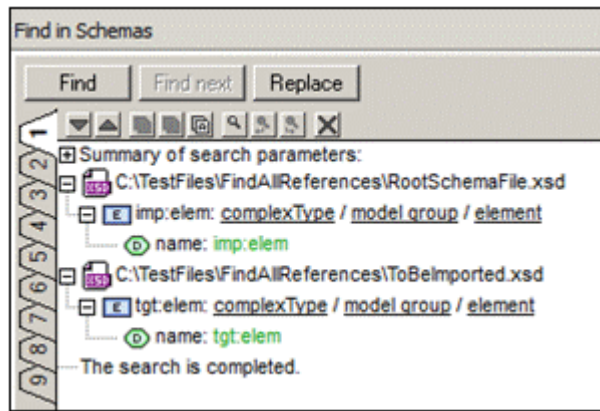
Named global components of XML Schemas can be found and renamed in a selected file and in all schema files related to the selected file. Named global components are all global components except: Include, Import, Redefine, Annotation, Comment, and PI components

The process works as follows:

1. In Schema Overview, the global component to be found or renamed is selected.
2. In the context menu that pops up on right-clicking the selected component, select the required command (**Find All References** or **Rename with All References**).



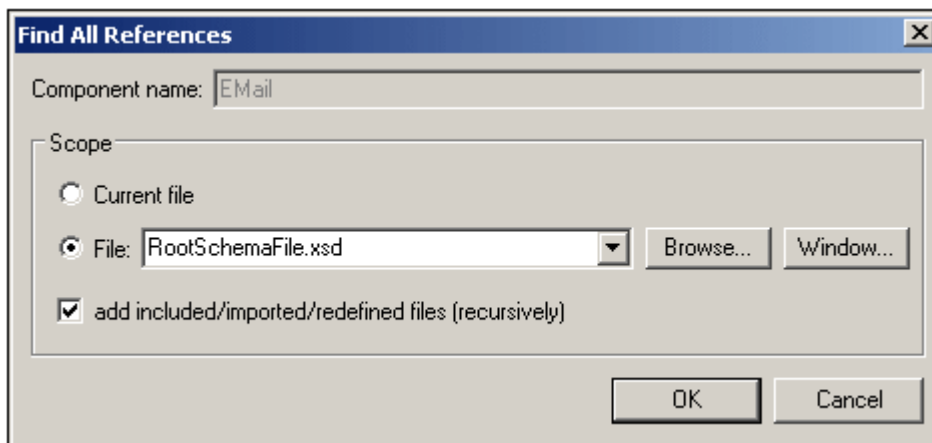
3. In the dialog that pops up, select the scope of the search (or rename operation). In the case of a Rename operation, enter the new name of the global component.
4. On clicking **OK**, the search results are displayed in the Find in Schemas window (*screenshot below*).



The locations of all files in which references to the global component are found are listed (see *screenshot above*). All renamed components that were found and renamed are also listed.

Find All References

To open the Find All References dialog (*screenshot below*), do the following: (i) Right-click the global component in Schema Overview, (ii) In the context menu that pops up select the **Find All References** command.

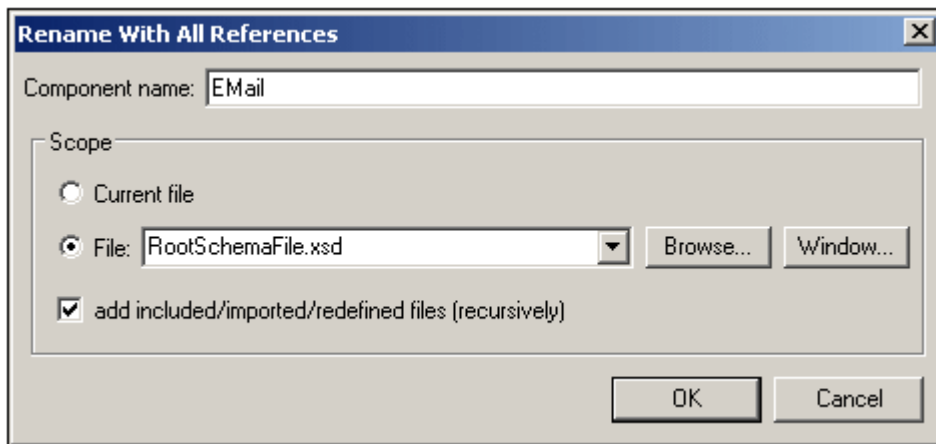


The global component name is displayed in the *Component Name* field, which is grayed out and cannot be edited. You can choose whether the search should be carried out in the current file or

in another file you can browse for (or select from a list of open files). You can also then specify whether related files (included, imported, redefined) should be searched, by checking the Add IIR Files check box at the bottom of the dialog.

Rename with All References

To rename a global component, right-click it and select **Rename with All References** from the context menu that pops up. This pops up the Rename with All References dialog (*screenshot below*).



The new name you wish to give the selected global component must be entered in the *Component Name* text field. You can choose whether the search and renaming should be carried out in the current file or in another file you can browse for (or select from a list of open files). You can also then specify whether related files (included, imported, redefined) should be searched, by checking the Add IIR Files check box at the bottom of the dialog.

7 XSLT and XQuery

XMLSpy provides editing support for XSLT and XQuery documents, has built-in engines for transforming with XSLT and executing XQuery documents, as well as powerful debuggers and profilers. This section provides an overview of the XSLT and XQuery functionality available in XMLSpy. It is organized into the following sections:

- [XSLT](#)
- [XQuery](#)
- XSLT and XQuery Debugger
- XSLT/XQuery Profiler

Additional and more detailed information about commands is in the descriptions of the [relevant menu commands](#) (in the User Reference section). For more information on editing support, also see the [Editing Views](#) section and the [XML](#) section.

7.1 XSLT

Altova website:  [XSLT Editor](#)

This section on XSLT is organized into the following sections:

- [Editing XSLT documents](#): describes the editing support for XSLT documents in XMLSpy
- [XSLT Processing](#): shows the various ways in which XSLT transformations can be carried out in the XMLSpy GUI using engines of your choice. This section also explains important XSLT settings in XMLSpy.
- [XSL Outline](#): describes the XSL Outline and XSL Info Windows, which together provide a powerful way to view, navigate, and manage a collection of XSLT files.

XPath Evaluation

When an XML document is active, you can use the [XPath Window](#) to evaluate XPath expressions. This is a very useful feature to quickly check how an XPath expression will be evaluated. Type in an XPath expression and specify whether it should be evaluated relative to the document root or to a selected context node in the XML document. The result of the evaluation will be displayed immediately in the XPath Window. How to use the XPath Window is described in the section [Introduction | XPath Window](#).

XSLT Profiler and Debugger

XMLSpy also contains an [XSLT Profiler](#) and [XSLT Debugger](#) to help you create correct and efficient XSLT stylesheets faster. These two features are described in the section [XSLT and XQuery](#).

Additional XSLT features

Additional and more detailed information about the various features described in this section is in the descriptions of the [relevant menu commands](#) (in the User Reference section).

Altova XSLT Engines

For details about how the Altova XSLT 1.0 and 2.0 Engines are implemented, see [Engine Information in the Appendices](#).

AltovaXML for command line and batch processing

The XMLSpy GUI enables batch processing via the projects functionality. However, if you are looking for more flexibility, you should try [Altova's free AltovaXML product](#), which contains the Altova XSLT 1.0 and 2.0 Engines that are built into XMLSpy. AltovaXML is ideal if you wish to perform XSLT transformations from the command line, or batch processing.

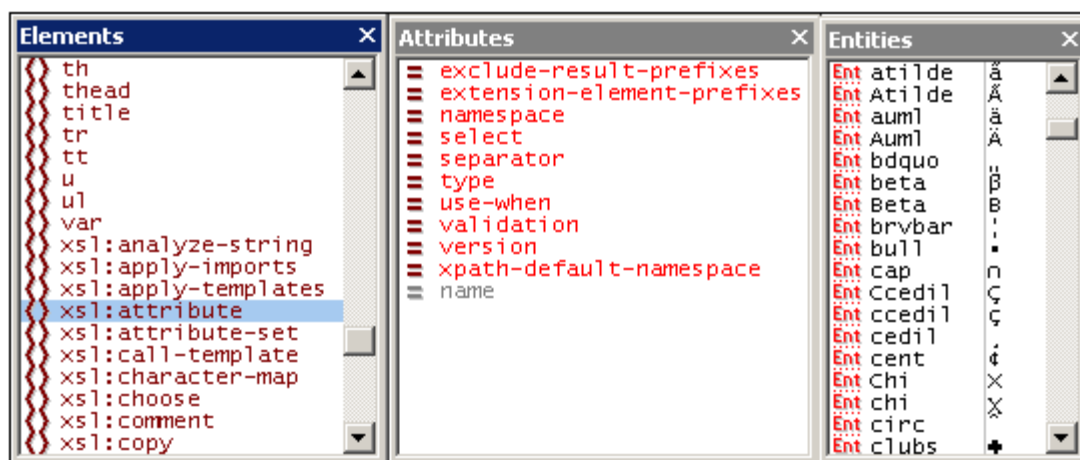
7.1.1 XSLT Documents

XSLT 1.0 and 2.0 documents can be edited in [Text View](#) and [Grid View](#), and are edited like any other XML document in [Text View](#) and [Grid View](#). The default view in which an XSLT document is opened can be set in the File Types tab of the Options dialog.

Entry helpers

Entry helpers are available for elements, attributes, and entities. Information about the items

displayed in the entry helpers is built into XMLSpy, and is not dependent on references contained in the XSLT document.



The following points should be noted:

1. If a new XSLT document is created via the Create a New Document dialog (**File | New**), then the appropriate XSLT elements and attributes (XSLT 1.0 or XSLT 2.0, depending upon which document type was created) are loaded into the entry helpers. Additionally, HTML elements and attributes are loaded, as well as the HTML 4.0 entity sets, [Latin-1](#), [special characters](#), and [symbols](#).
2. If an XML document is created via the Create a New Document dialog (**File | New**) and given XSLT content, no entry helper items are available except for XML character entities.
3. If an XSLT document is opened that was created as an XSLT document via the Create a New Document dialog (**File | New**), then the entity helpers will be as in Point 1 above.
4. If an XSLT document is opened that was **not** created as an XSLT document via the Create a New Document dialog (**File | New**), then the entity helpers will be as in Point 1 above. Additionally, XSL-FO elements and attributes will be listed in the Text View entry helpers.
5. The prefixes of elements in the Elements entry helper are as follows and are invariable:
`xsl:` prefix for XSLT elements; no prefix for HTML elements; `fo:` prefix for XSL-FO elements. Consequently, in order to use the entry helpers, the namespace declarations in the XSLT document must define prefixes that match the built-in prefixes shown in the entry helpers.

Auto-completion

In Text View, auto-completion is available in a pop-up as you type, with the first item in the pop-up list being highlighted that matches the typed text. When an element is being typed, a list of elements pops up with the first nearest match in alphabetical order being highlighted. Similarly, when an attribute is being typed in, a list of applicable attributes pops up. The items in the list are determined according to the rules described in the previous section on entry helpers.

XPath intelligent editing

At locations in the XSLT document where XPath expressions can be entered (for example, the value of a `select` attribute), intelligent XPath editing is available. XPath functions and XPath axes become available in popup as you type. Additionally, if an XML file has been assigned in the [Info window](#), the elements and attributes of the XML file will also be available in the popup.

Validating XSLT documents

The XSLT document can be validated against the XSLT schema built into XMLSpy (click **XML | Validate (F8)**). The correct built-in schema is automatically selected according to whether the XSLT document is XSLT 1.0 or XSLT 2.0 (specified in the `version` attribute of the `xsl:stylesheet` element).

7.1.2 XSLT Processing

In the XMLSpy GUI, two types of XSLT transformation are available:

- The **XSL/XQuery | XSL Transformation (F10)** command is used for straightforward XML transformations with an XSLT stylesheet to result formats specified and described in the stylesheets.
- The **XSL/XQuery | XSL-FO Transformation** command is used: (i) for transformations of XML to FO to PDF in two steps, and (ii) for one-step transformations of FO to PDF.

Specifying the XSLT processor for the transformation

The XSLT engine that will be used for transformations is specified in the [XSL tab of the Options dialog](#) (screenshot below).

☒ Built-in XSLT engine (Important: the built-in XSLT engine is always used for XSLT debugging)
☐ Microsoft® XML Parser (MSXML): ☐ v3.0 ☐ v4.0 ☐ v6.0 ☒ Choose version automatically
☐ External XSL transformation program:

 Please enter the command line for executing an external XSL transformation program in the form:
 Program.exe %1 %2 %3
 where %1 will be replaced with the XML input file name, %2 with the output file name and %3 (optional) with XSL style-sheet file name. Feel free to add any other parameters that are required by the external program.
☒ Show external program output in Messages window after transformation
☒ Show external program error output in Messages window after transformation
 Default file extension of output file: ☒ Reuse output window
☒ Use file extension from <xsl:output method=""> attribute if provided
 Please enter path to XSL-FO transformation engine (if using FOP enter path to fop.bat):

☒ Use XSLT engine selected above to perform XSLT part and then XSL-FO engine for FO part
☐ Use XSL-FO engine for both XSLT and FO parts of transformation

The available options are explained in the [User Reference](#) section. The engine specified in the XSL tab will be used for all XSLT transformations. Note that for the XSL:FO transformation, an additional XSLT engine option is available: the XSLT engine that is packaged with some FO processors. To select this option, select the corresponding radio button at the bottom of the XSL tab (see screenshot above).

Specifying the FO processor

The FO processor that will be used for transformations of FO to PDF is specified in the text box at the bottom of the [XSL tab of the Options dialog](#) (screenshot above).

XSLT 1.0 and 2.0 and Altova's XSLT engines

The XSLT version of a stylesheet is specified in the `version` attribute of the `xsl:stylesheet` (or `xsl:transform`) element. XMLSpy contains the built-in Altova XSLT 1.0 and Altova XSLT 2.0 engines, and the appropriate engine is selected according to the value of the `version` attribute (1.0 or 2.0).

XSLT Transformation

The **XSLT Transformation (F8)** command can be used in the following scenarios:

- To transform an XML document that is active in the GUI and has an XSLT document [assigned](#) to it. If no XSLT document is assigned, you are prompted to make an assignment when you click the **XSLT Transformation (F8)** command.
- To transform an XSLT document that is active in the GUI. On clicking the **XSLT Transformation (F8)** command, you are prompted for the XML file you wish to process with the active XSLT stylesheet.
- To transform project folders and files. Right-click the project folder or file and select the command.

XSL:FO Transformation

The **XSL:FO Transformation** command can be used in the following scenarios:

- To transform an XML document that is active in the GUI and has an XSLT document [assigned](#) to it. The XML document will first be transformed to FO using the specified XSLT engine. The FO document will then be processed with the specified FO processor to produce the PDF output. If no XSLT document is assigned, you are prompted to make an assignment when you click the **XSL:FO Transformation** command.
- To transform an FO document to PDF using the specified FO processor.
- To transform an XSLT document that is active in the GUI. On clicking the **XSL:FO Transformation** command, you are prompted for the XML file you wish to process with the active XSLT stylesheet.
- To transform project folders and files. Right-click the project folder or file and select the command.

For a description of the options in the [XSL:FO output dialog](#), see the [User Reference section](#).

Parameters for XSLT

If you are using the Altova XSLT engines, XSLT parameters can be stored in a convenient GUI dialog. All the stored parameters are passed to the XSLT document each time you transform. For more information, see the description of the [XSLT Parameters / XQuery Variable](#) command.

Batch processing with AltovaXML

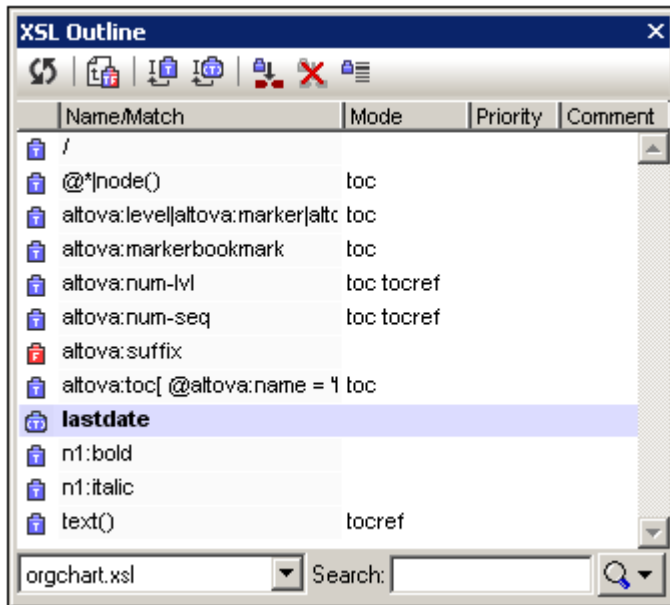
AltovaXML is a free standalone application that contains Altova's XML validator, XSLT engines, and XQuery engine. It can be used from the command line, via a COM interface, in Java programs, and in .NET applications to validate XML documents, transform XML documents using XSLT 1.0 and 2.0 stylesheets, and execute XQuery documents.

XSLT transformation tasks can therefore be automated with the use of Altova XML. For example, you can create a batch file that calls AltovaXML to transform a set of documents. See the [Altova XML documentation](#) for details.

7.1.3 XSL Outline

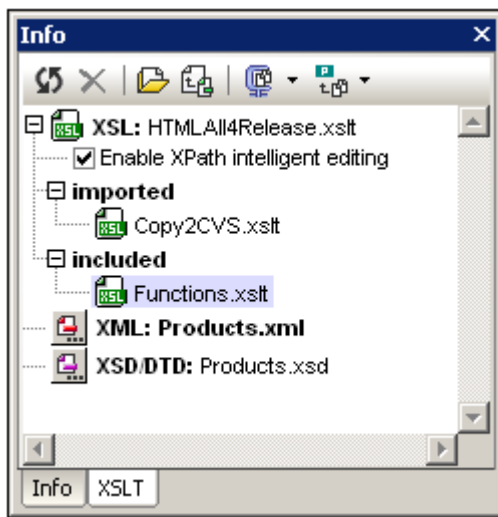
When an XSLT document is the active document in XMLSpy, information about the structure of the document is displayed in the [XSL Outline window](#) and information about the files related to the active XSLT document is displayed in the [XSLT tab of the Info Window](#) (which is displayed only when an XSLT document is the active document in XMLSpy). Additionally, via these two windows, a number of commands are available that facilitate editing the XSLT document and managing files related to it.

In the [XSL Outline window](#) (screenshot below), you can do the following:



- View the templates and functions in the active XSLT document and in all imported and included XSLT documents.
- Sort the templates and functions on the basis of their names or match expressions, mode, priority, or comments.
- Search for specific templates on the basis of their names/expressions.
- Use the XSL Outline to navigate to the corresponding template in the XSLT document.
- Quickly insert calls to named templates.
- Set a selected named template as the entry point for transformations.

In the [XSLT tab of the Info Window](#) (screenshot below), you can do the following:

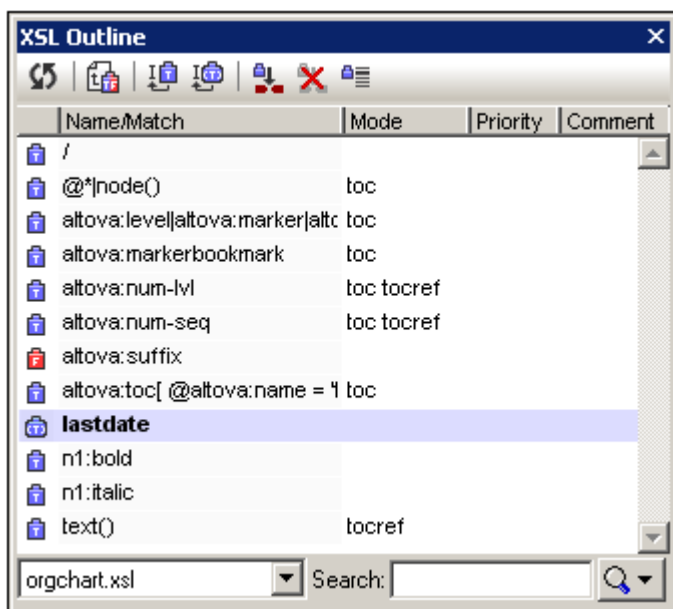


- View information about all the files related to the active XSLT document, such as the locations of imported and included files.
- Set an XML file for transformation with the active XSLT document. Also, the schema (XSD/DTD) file can be set for validating the selected XML file.
- Open a related file from within the Info Window.
- Quickly organize all related files into XMLSpy projects.
- Zip all related files to a user-defined location.

The [XSL Outline window](#) and the [XSLT tab of the Info Window](#) are described in detail in the sub-sections of this section.

XSL Outline Window

In the XSL Outline Window (*screenshot below*), all templates and functions in the active XSLT document are listed. Templates are indicated with blue icons ( templates without a parameter; and  templates containing parameters). Functions are indicated with a red  icon. In the combo box in the bottom left-hand of the window, you can select whether the templates and functions listed are from: (i) only the active XSLT document (as in the screenshot below), or (ii) the active XSLT document and all included and imported stylesheets.



There are two types of templates: (i) named templates, and (ii) templates that match an XPath expression. Each template is listed with:

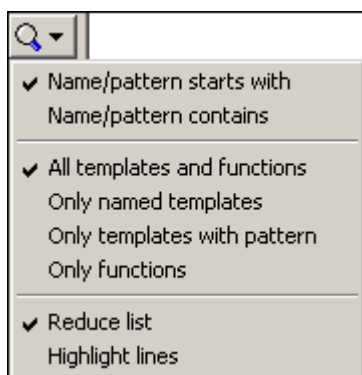
- Its name (if the template has a `name` attribute) and/or XPath expression (if the template has a `match` attribute). If the template has both, a `name` and a `match` attribute, then both are listed, with the value of the `name` attribute first: `namevalue, matchvalue` (see *the template named bold in the screenshot above*).
- Its mode, if any;
- Its priority, if any;
- The comment that directly precedes the template or function, if any.

Functions in the stylesheet are listed by their names. Functions have neither mode nor priority.

Operations

The following operations can be performed in the XSL Outline Window:

- *Filtering*: The list displayed in the window can be filtered to show one of the following: (i) all templates and functions (the default setting each time XMLSpy is started); (ii) named templates only; (iii) XPath-expression templates only; (iv) functions only. To select the required filter, click the dropdown arrow to the right of the Search box at bottom right of the window (*screenshot below*), and select the required filter (the second group of commands in the menu). The selected filter is applied immediately and applies from this moment onwards till it is modified or till XMLSpy is closed.



- **Sorting and locating:** Each column can be sorted alphabetically by clicking the column header. Each subsequent click reverses the previous sorting order. After a column has been sorted in this way, if you select any item in the list and then quickly type in a term from the sorted column, the first item in the list that contains that term will be highlighted. In this way, you can quickly go to templates of a particular name/ expression, mode, or priority.
- **Searching:** Enter in the Search box (at bottom right) the name or XPath expression for which you wish to search. The search results are displayed as you type. The following search options are available in the dropdown list of the Search box (*screenshot above*): (i) whether the name or expression either starts with or contains the search term (the first group of commands in the menu); the starts-with option is the default each time XMLSpy is started; (ii) whether the search results should be displayed as a reduced list or be highlighted (the third group of commands in the menu); the reduced-list option is the default each time XMLSpy is opened. These selections are applied immediately and remain in effect till changed or till XMLSpy is closed.
- **Reloading:** After the stylesheet has been modified, click the **Reload** icon  in the window's toolbar to update the XSL outline.
- **Go to item:** When a template or function is selected in the XSL Outline window, clicking the **Go to Item** icon  in the window's toolbar highlights the template or function in the document in Design View. Alternatively, double-click an entry to go to it.
- **Named template actions:** Two groups of actions can be carried out involving named templates: (i) Calls to the named template (with `xsl:call-template`) can be inserted in the stylesheet at the cursor insertion point; and (ii) A named template can be set as the entry point for a transformation. The commands for these actions are carried out via icons in the toolbar and are described below.

Named templates

When a named template is selected, one or more commands in the window's toolbar relating to named templates become enabled (*screenshot below*).



The commands in the toolbar (*screenshot above*) are, from left to right:

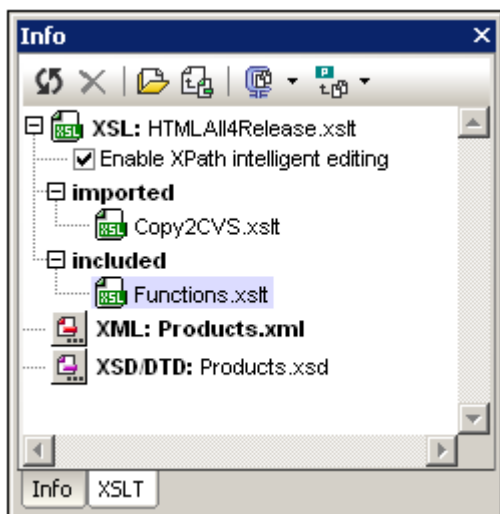
- **Insert `xsl:call-template`:** This command becomes active when a named template is selected in the XSL Outline window. The command inserts an `xsl:call-template` element at the cursor insertion point in the stylesheet. The `name` attribute of the `xsl:call-template` element that is inserted in the stylesheet is given a value that is the value of the `name` attribute of the selected named template. This makes the `xsl:call-template` a call to the selected named template.
- **Insert `xsl:call-template` with `param`:** This command becomes active when a named

template having one or more `xsl:param` child elements is selected in the XSL Outline window. As with the **Insert xsl:call-template** command, the command inserts an `xsl:call-template` element, but in this case with a corresponding `xsl:with-param` child element for every `xsl:param` child element of the selected named template. The names of the inserted `xsl:call-template` and its `xsl:with-param` child elements correspond to the names of the selected named template and its `xsl:param` children.

- *Set the selected named template as entry point for transformation:* When a named template is set as the entry point for a transformation, transformations executed in XMLSpy start at this named template. In the XSL Outline Window, such a named template is indicated in boldface (see screenshot at the start of this section).
- *Clear named template as entry point for transformation:* Becomes active once a named template has been set as the entry point for transformations.
- *Jump to the named template selected as the entry point for transformations:* Becomes active once a named template has been set as the entry point for transformations. When the focus in the XSL Outline window is at some other point than the named template set as the entry point for transformations, clicking this icon highlights the named template in the XSL Outline window, thus making access to it faster.

Info Window

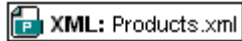
The XSLT tab of the Info Window, which is displayed only when an XSLT document is the active document in XMLSpy, displays all the imported and included XSLT files related to the active XSLT document, and enables the selection of an XML file that can be used as the source XML document on which the XSLT document is used for the transformation.



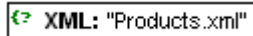
The following files are displayed in the XSLT tab of the Info Window:

- **XSLT files:** All imported and included XSLT files are listed (see screenshot above). The location of each file is displayed in a pop-up when the mouse cursor is placed over the file. Double-clicking an imported or included file, or selecting it and then clicking the **Open** icon in the Info Window toolbar, opens the file in a new window. The **Go to Include/Import Location** icon in the toolbar highlights the include/import declaration in the active XSLT document.
- **XML file:** An XML file can be assigned to the active XSLT stylesheet for transformations. The location of the assigned XML file is displayed in a pop-up when the mouse cursor is placed over the file. If an XML file is specified and the menu

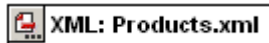
command **XSL/XQuery | XSL Transformation (F10)** is clicked, a transformation is executed on the defined XML file using the active XSLT document as the stylesheet. The XML file can be selected by clicking the **XML** icon and browsing; the selected file is displayed in bold face. Alternatively, the XML file can be assigned via the Project Properties dialog (*XSL transformation of XSL files*) or via a processing instruction in the XSLT document of the form: `<?altova_samplexml "Products.xml"?>`. In each case, the XML file will be shown in the Info Window with the relevant icon:



assigned via the Project Properties dialog



assigned via a processing instruction in the XSLT document



assigned by clicking the XML icon and browsing for the required file; entry is in bold font face

In the event that more than one of the above assignments exists, the selection priority is: (i) project; (ii) processing instruction; (iii) browsed by user. The XML file can be opened by double-clicking it or by selecting it and clicking the **Open** toolbar icon.

- **XSD/DTD file:** If the selected XML file has a reference to a schema (XML Schema or DTD), then this schema file is displayed in the XSD/DTD entry. Alternatively, just as with the XML file, the schema file can be selected via the Project Properties dialog (*Validation*) or by clicking the **XSD/DTD** icon and browsing for the required schema file. If the schema file is selected via the Projects Properties dialog, a Projects icon is displayed next to the entry, otherwise the clickable **XSD/DTD** icon is displayed with the file entry either in a normal font face (when the schema is referenced from the XML file) or bold font face (schema browsed for by the user via the **XSD/DTD** icon). Should the schema file be assigned via more than one method, then the order of priority is as follows: (i) project; (ii) browsed by user; (iii) reference in XML document. The location of the assigned XSD file is displayed in a pop-up when the mouse cursor is placed over the file. The schema file can be opened by double-clicking it or by selecting it and clicking the **Open** toolbar icon.

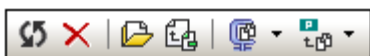
Note: If an XML or XSD/DTD file is selected via the Project Properties dialog, then to clear this selection, you must go to the Project Properties dialog and clear the setting there. If the selection has been made by browsing via the **XML** or **XSD/DTD** icons, then to clear this setting, select the file and click the **Clear** icon in the Info Window toolbar.

Options

XPath intelligent editing: If an XML file has been assigned, the structure of the XML document is known and [intelligent XPath editing](#) will extend to elements and attributes. At locations in the XSLT document where an XPath expression can be entered, available elements and attributes will be shown in a popup. This option is switched on by default. To disable XPath intelligent editing, uncheck the check box. The setting is saved for each XSLT file separately when the file is closed, and will be used each time the file is opened.

Toolbar icons

The Info Window toolbar icons (*screenshot below*) are, from left to right:



- **Reload info:** Updates the Info Window to reflect modifications made in the XSLT

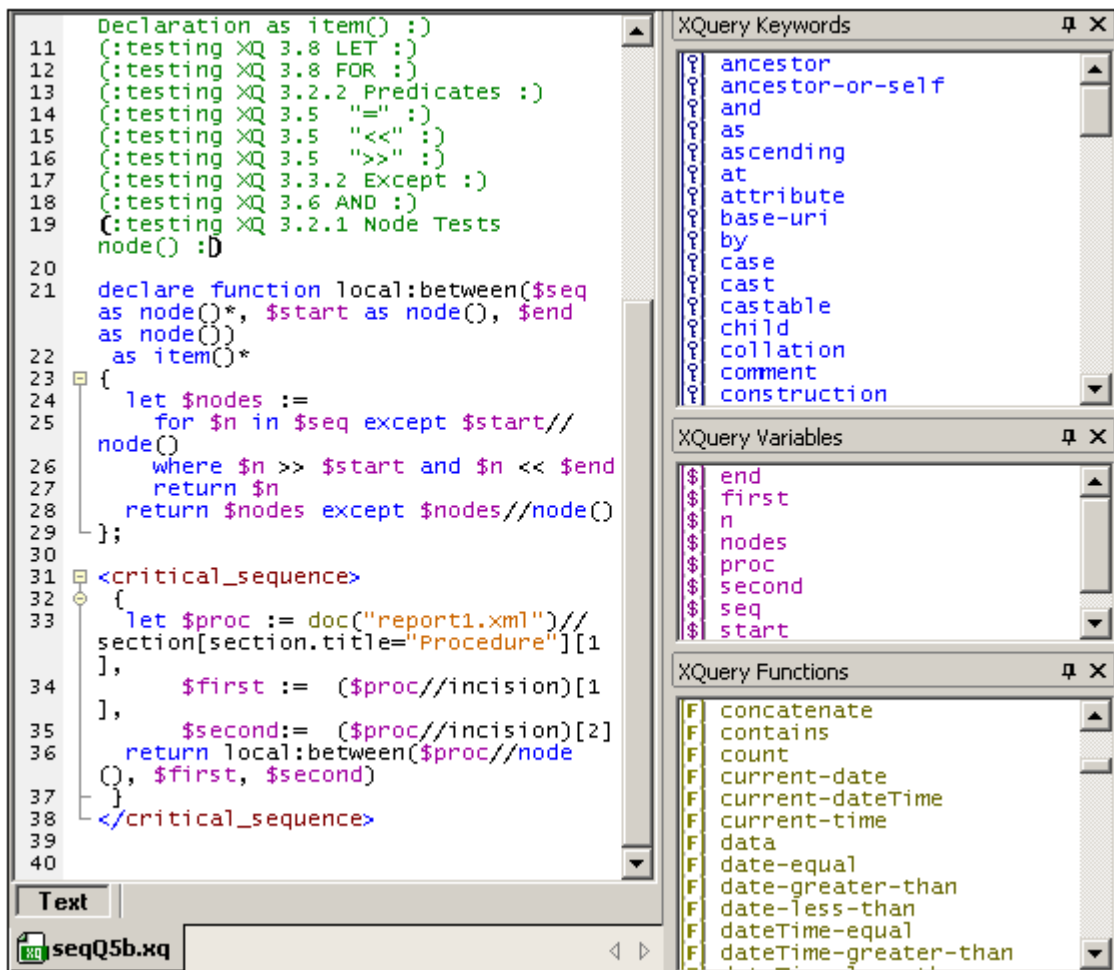
document.

- *Clear XML/XSD assignment:* Clears an XML or XSD/DTD assignment made by the user by browsing via the XML or XSD/DTD icons, respectively. Select the file to clear and then click this icon.
- *Open document:* Opens the selected document.
- *Go to import/include location:* When an imported or included file is selected, clicking this icon highlights the relevant import or include declaration in the XSLT document.
- *Zip all local documents:* Zips all the documents listed in the Info Window to a user-defined location. Alternatively, only the selected documents can be zipped; do this by selecting, in the dropdown menu of this icon, the command **Zip selected local documents**.
- *Add all files to projects:* Adds all files to the current projects. Alternatively, only the selected documents can be added; do this by selecting, in the dropdown menu of this icon, the command **Add selected files to project**.

7.2 XQuery

Altova website:  [XQuery Editor](#)

XQuery documents can be edited in Text View. The Entry Helpers, syntax coloring, and intelligent editing are different than for XML documents (see screenshot; *line numbering and folding margins in Enterprise and Professional Editions only*). We call this mode of Text View its XQuery Mode. In addition, you can validate your XQuery document in Text View and execute the code in an XQuery document (with an optional XML file if required) using the built-in Altova XQuery Engine.



Please note: XQuery files can be edited only in Text View. No other views of XQuery files are available.

XQuery Profiler and Debugger

XMLSpy also contains an [XQuery Profiler](#) and [XQuery Debugger](#) to help you create correct XQuery documents faster. These two features are described in the section [XSLT and XQuery](#).

Altova XQuery Engine

For details about how the Altova XQuery Engine is implemented and will process XQuery files,

see [XQuery Engine Implementation](#).

AltovaXML for command line and batch processing

The XMLSpy GUI enables batch processing via the projects functionality. However, if you are looking for more flexibility, you should try [Altova's free AltovaXML product](#), which contains the Altova XQuery Engine that is built into XMLSpy. AltovaXML is ideal if you wish to perform XQuery executions from the command line, or batch processing.

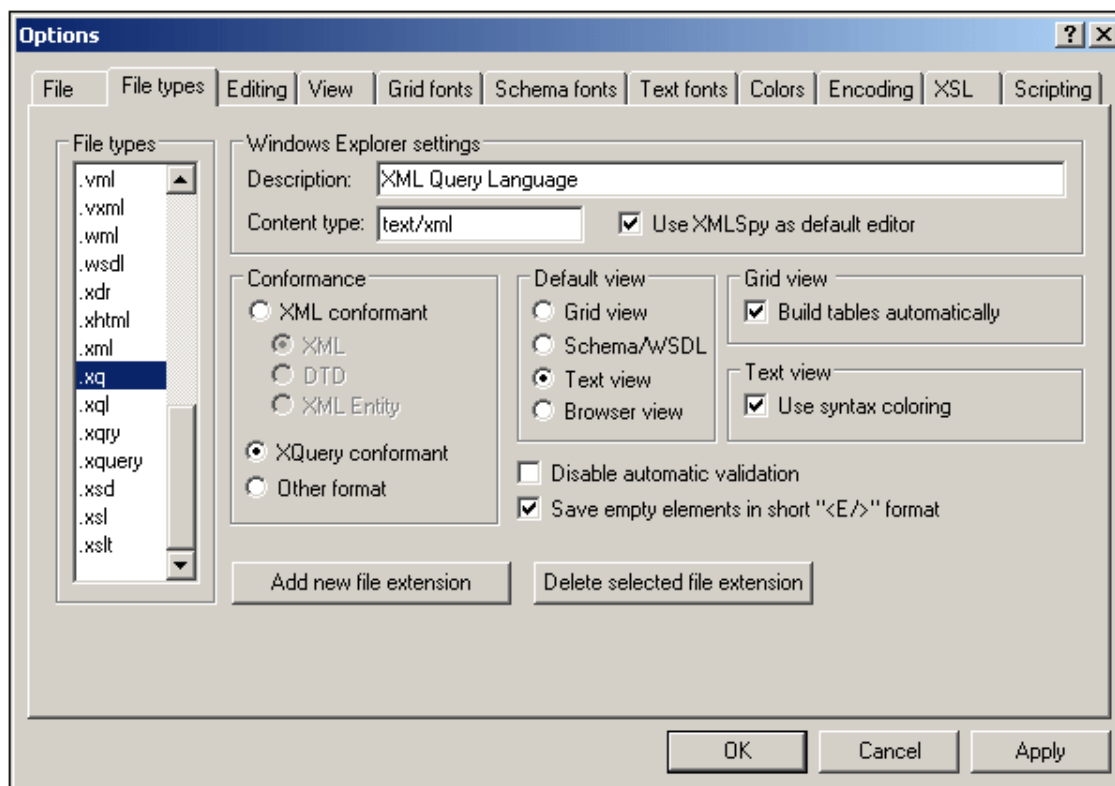
7.2.1 XQuery Documents

An XQuery document is opened automatically in XQuery Mode of Text View if it is XQuery conformant. Files that have the file extension `.xq`, `.xql`, and `.xquery` are pre-defined in XMLSpy as being XQuery conformant.

Setting additional file extensions to be XQuery conformant

To set additional file extensions to be XQuery conformant:

1. Select **Tools | Options**. The Options dialog appears (see *screenshot*).
2. Select the **File types** tab.
3. Click Add new file extension to add the new file extension to the list of file types.
4. Under **Conformance**, select **XQuery conformant**.

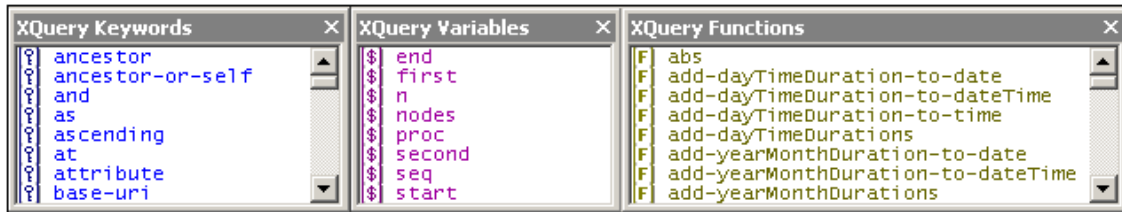


You should also make the following **Windows Explorer settings** in this dialog:

- **Description:** XML Query Language
- **Content type:** text/xml
- If you wish to use XMLSpy as the default editor for XQuery files, activate the **Use XMLSpy as default editor** check box.

7.2.2 XQuery Entry Helpers

There are three Entry Helpers in the XQuery Mode of Text View: XQuery Keywords (blue), XQuery Variables (purple), and XQuery Functions (olive).



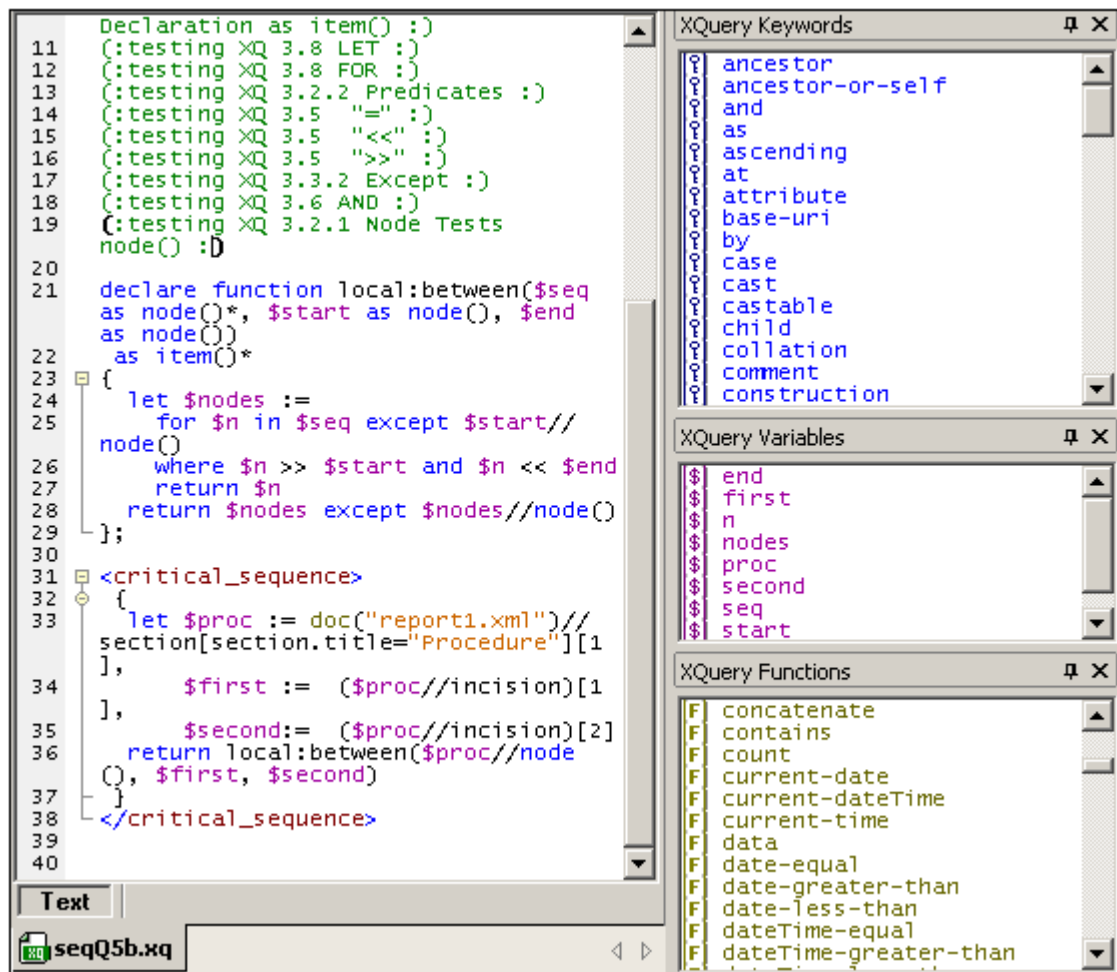
Note the following points:

- The color of items in the three Entry Helpers are different and correspond to the syntax coloring used in the text. These colors cannot be changed.
- The listed keywords and functions are those supported by the Altova XQuery Engine.
- The variables are defined in the XQuery document itself. When a \$ and a character are entered in Text View, the character is entered in the Variables Entry Helper (unless a variable consisting of exactly that character exists). As soon as a variable name that is being entered matches a variable name that already exists, the newly entered variable name disappears from the Entry Helper.
- To navigate in any Entry Helper, click an item in the Entry Helper, and then use either the scrollbar, mouse wheel, or page-down and page-up to move up and down the list.

To insert any of the items listed in the Entry Helpers into the document, place the cursor at the required insertion point and double-click the item. In XQuery, some character strings represent both a keyword and a function (`empty`, `unordered`, and `except`). These strings are always entered as keywords (in blue)—even if you select the function of that name in the Functions Entry Helper. When a function appears in blue, it can be distinguished by the parentheses that follow the function name.

7.2.3 XQuery Syntax Coloring

An XQuery document can consist of XQuery code as well as XML code. The default syntax coloring for the XQuery code is described in this section. The syntax coloring for XML code in an XQuery document is the same as that used for regular XML documents. All syntax coloring (for both XQuery code and XML code) is set in the Text Fonts tab of the Options dialog (**Tools | Options**). Note that XQuery code can be contained in XML elements by enclosing the XQuery code in curly braces { } (see screenshot for example).



In XQuery code in the XQuery Mode of Text View, the following default syntax coloring is used:

- (: Comments, including 'smiley' delimiters, are in green :)
- XQuery Keywords are in blue: **keyword**
- XQuery Variables, including the dollar sign, are in purple: **\$start**
- XQuery Functions, but **not** their parentheses, are in olive: **function()**
- Strings are in orange: **"Procedure"**
- All other text, such as path expressions, is black (*shown underlined below*). So:

```
for $s in doc("report1.xml") //section[ section.title =
"Procedure"]
return ($s//incision)[2]/instrument
```

You can change these default colors and other font properties in the Text Fonts tab of the Options dialog (**Tools | Options**).

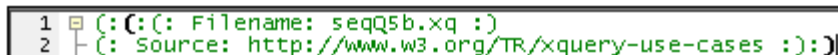
Please note: In the above screenshot, one pair of colored parentheses for a comment is displayed black and bold. This is because of the bracket-matching feature (see [XQuery Intelligent Editing](#)).

7.2.4 XQuery Intelligent Editing

The XQuery Mode of Text View provides the following intelligent editing features.

Bracket-matching

The bracket-matching feature highlights the opening and closing brackets of a pair of brackets, enabling you to clearly see the contents of a pair of brackets. This is particularly useful when brackets are nested, as in XQuery comments (see *screenshot below*).



```
1  (: (: (: Filename: seqQ5b.xq :))
2  (: Source: http://www.w3.org/TR/xquery-use-cases :):)
```

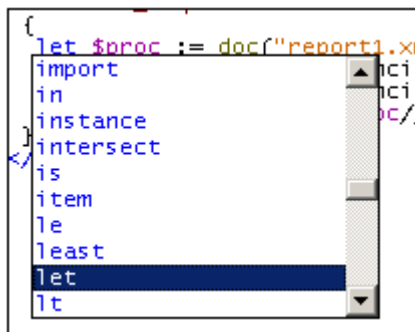
Bracket-matching is activated when the cursor is placed either immediately before or immediately after a bracket (either opening or closing). That bracket is highlighted (bold black) together with its corresponding bracket. Notice the cursor position in the screenshot above.

Bracket-matching is enabled for round parentheses `()`, square brackets `[]`, and curly braces `{}`. The exception is angular brackets `<>`, which are used for XML tags.

Please note: When you place the cursor just inside a start or end bracket, both brackets are highlighted. Pressing **Ctrl+E** moves the cursor to the other member of the pair. Pressing **Ctrl+E** repeatedly enables you to switch between the start and end brackets. This is another aid to quickly navigating your document.

Keywords

XQuery keywords are instructions used in query expressions, and they are displayed in blue. You select a keyword by placing the cursor inside a keyword, or immediately before or after it. With a keyword selected, pressing **Ctrl+Space** causes a complete list of keywords to be displayed in a pop-up menu. You can scroll through the list and double-click a keyword you wish to have replace the selected keyword.



```
{
let $proc := doc("report1.x")
import
in
instance
intersect
is
item
le
least
let
lt
}
```

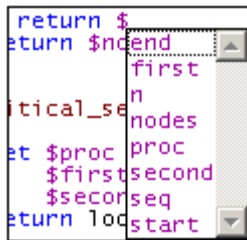
In the screenshot above, the cursor was placed in the `let` keyword. Double-clicking a keyword from the list causes it to replace the `let` keyword.

Variables

Names of variables are prefixed with the `$` sign, and they are displayed in purple. This mechanism of the intelligent editing feature is similar to that for keywords. There are two ways to access the pop-up list of all variables in a document:

- After typing a `$` character, press **Ctrl+Space**
- Select a variable and press **Ctrl+Space**. (A variable is selected when you place the cursor immediately after the `$` character, or within the name of a variable, or

immediately after the name of a variable.)



To insert a variable after the \$ character (when typing), or to replace a selected variable, double-click the variable you want in the pop-up menu.

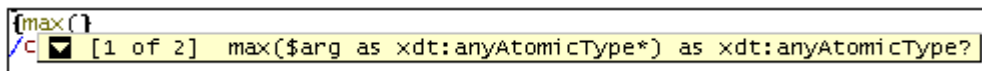
Functions

Just as with keywords and variables, a pop-up menu of built-in functions is displayed when you select a function (displayed in olive) and press **Ctrl+Space**. (A function is selected when you place the cursor within a function name, or immediately before or after a function name. The cursor must not be placed between the parentheses that follow the function's name.)

Double-clicking a function name in the pop-up menu replaces the selected function name with the function from the pop-up menu.

To display a tip containing the signature of a function (*screenshot below*), place the cursor immediately after the opening parenthesis and press **Ctrl+Space**.

Please note: The signature can be displayed only for standard XQuery functions.



The downward-pointing arrowhead indicates that there is more than one function with the same name but with different arguments or return types. Click on the signature to display the signature of the next function (if available); click repeatedly to cycle through all the functions with that name. Alternatively, you can use the **Ctrl+Shift+Up** or **Ctrl+Shift+Down** key-combinations to move through a sequence.

Visual Guides

Text folding is enabled on XQuery curly braces, XQuery comments, XML elements, and XML comments.

7.2.5 XQuery Validation and Execution

Validating XQuery documents

To validate an XQuery document:

1. Make the XQuery document the active document.
2. Select **XML | Validate**, or press the **F8** key, or click the  toolbar icon.

The document will be validated for correct XQuery syntax.

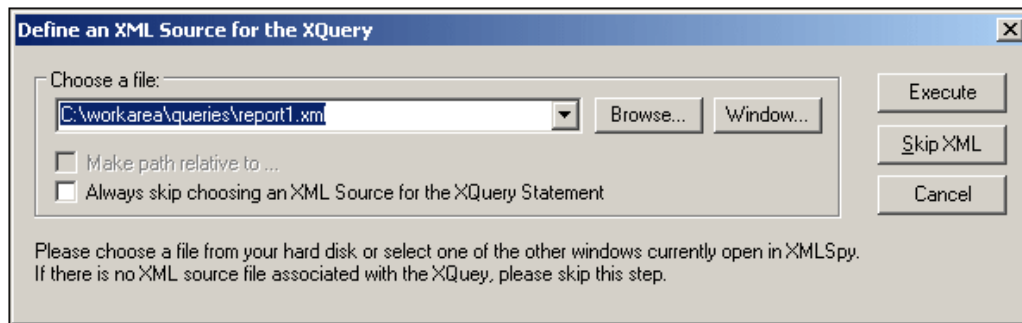
Executing XQuery documents

XQuery documents are executed within XMLSpy using the built-in XQuery 1.0 engine. The output is displayed in a window in XMLSpy.

Typically, an XQuery document is not associated with any single XML document. This is because XQuery expressions can select any number of XML documents with the `doc()` function. In XMLSpy, however, before executing individual XQuery documents you can select a source XML document for the execution. In such cases, the document node of the selected XML source is the starting context item available at the root level of the XQuery document. Paths that begin with a leading slash are resolved with this document node as its context item.

To execute an XQuery document:

1. Make the XQuery document the active document.
2. Select **XSL/XQuery | XQuery Execution** or click the  toolbar icon. This opens the Define an XML Source for the XQuery dialog.



3. Do one of the following:
 - To select an XML file, use either the **Browse** button or the **Window** button (which lists files that are open in XMLSpy and that are in XMLSpy projects). Select an XML source if you wish to assign its document node as the context item for the root level of the XQuery document. Click **Execute**.
 - To skip this dialog click **Skip XML**.

The result document is generated as a temporary file that can be saved to any location with the desired file format and extension.

XQuery Variables

If you are using the Altova XQuery engine, XQuery variables can be stored in a convenient GUI dialog. All the stored variables are passed to the XQuery document each time you execute an XQuery document via XMLSpy. For more information, see the description of the [XSLT Parameters / XQuery Variable](#) command.

Altova XQuery Engine

For details about how the Altova XQuery Engine is implemented and will process XQuery files, see [XQuery Engine Implementation](#).

7.2.6 XQuery and XML Databases

An XQuery document can be used to query an XML database (XML DB). Currently this XQuery functionality is supported only for IBM DB2 databases. The mechanism for querying an XML DB using XQuery essentially involves: (i) indicating to the XQuery engine that XML in a DB is to be queried—as opposed to XML in an XML document; and (ii) accessing the XML data in the DB.

The steps for implementing this mechanism are as follows and are described in detail below:

1. [Set up the XQuery document](#) to query the XML DB by inserting the `XQUERY` keyword at

- the start of the document.
2. For the active XQuery document, [enable DB support](#) (via the Info window) and [connect to the DB](#) (using the Quick Connect dialog).
3. In the XQuery document, insert [DB-specific XQuery extensions](#) so as to access the DB data and make it available for XQuery operations.
4. [Execute the XQuery](#) document in XMLSpy.

Setting up the XQuery document to query the XML DB

To set up the XQuery document to query an XML DB, open the XQuery document (or create a new XQuery document) and enter the keyword `XQUERY` (casing is irrelevant) at the start of the document (before the prolog); *see examples below*.

```
XQUERY (: Retrieve details of all customers :)
declare default element namespace "http://www.altova.com/xquery/databases/db2"
;
<a> { db2-fn:xmlcolumn( "CUSTOMER.INFO" ) } </a>
```

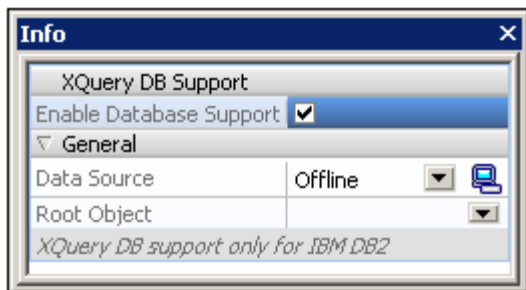
If the document uses the optional `xquery version` expression, the `XQUERY` keyword is still required:

```
XQUERY xquery version "1.0"; (: Retrieve details of all customers :)
declare default element namespace "http://
http://www.altova.com/xquery/databases/db2";
<a> { db2-fn:xmlcolumn( "CUSTOMER.INFO" ) } </a>
```

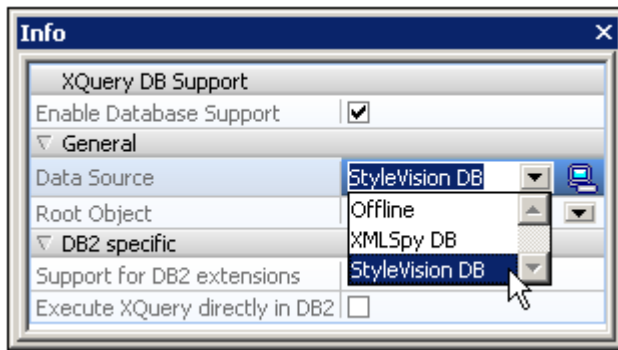
Note: XMLSpy's built-in XQuery Engine reads the `XQUERY` keyword as indicating that an XML DB is to be accessed. As a result, attempting to execute an XQuery document containing the `XQUERY` keyword on any XML document other than one contained in an XML DB will result in an error.

Enable DB support for XQuery and connect to the DB

DB support for an XQuery document is enabled by checking the Enable Database Support check box in the Info window (*screenshot below*). Note that DB Support must be enabled for each XQuery document separately and each time an XQuery document is opened afresh.



When you enable DB support in the Info window, a Quick Connect dialog pops up, which enables you to connect to a database. Currently, only IBM DB2 databases are supported. How to connect to a DB is described in the section, [Connecting to a Data Source](#). If connections to data sources already exists, then these are listed in the Data Sources combo box of the Info window (*screenshot below*), and one of these data sources can be selected as the data source for the active XQuery document. In the Info window, you can also select the root object from among those available in the Root Object combo box.



The Quick Connect dialog (which enables you to connect to a DB) can be accessed at any time by clicking the  icon in the Info window.

Note: When you close an XQuery document the connection to the DB is closed as well. If you subsequently re-open the XQuery document, you will also have to re-connect to the DB.

IBM DB2-specific XQuery language extensions

Two IBM DB2-specific functions can be used in XQuery documents to retrieve data from an IBM DB2 database:

- `db2-fn:xmlcolumn` retrieves an entire XML column without searching or filtering the column.
- `db2-fn:sqlquery` retrieves values based on an SQL `SELECT` statement

The XML data retrieved using these functions can then be operated on using standard XQuery constructs. *See examples below.*

db2-fn:xmlcolumn: The argument of the function is a case-sensitive string literal that identifies an XML column in a table. The string literal argument must be a qualified column name of type XML. The function returns all the XML data in the column as a sequence, without applying a search condition to it. In the following example, all the data of the `INFO` (XML) column of the `CUSTOMER` table is returned within a top-level `<newdocelement>` element:

```
XQUERY (: Retrieve details of all customers :)
declare default element namespace "http://www.altova.com/xquery/databases/db2"
;
<newdocelement> { db2-fn:xmlcolumn( "CUSTOMER.INFO" ) } </newdocelement>
```

The retrieved data can then be queried with XQuery constructs. In the example below, the XML data retrieved from the `INFO` (XML) column of the `CUSTOMER` table is filtered using an XQuery construct so that only the profiles of customers from Toronto are retrieved.

```
XQUERY (: Retrieve details of Toronto customers :)
declare default element namespace "http://www.altova.com/xquery/databases/db2"
;
<newdocelement> { db2-fn:xmlcolumn( "CUSTOMER.INFO" ) /customerinfo[ addr/city=
'Toronto' ] } </newdocelement>
```

Note: In the example above, the document element of the XML files in each cell is `customerinfo` and the root node of the XML sequence returned by `db2-fn:xmlcolumn` is considered to be an abstract node above the `customerinfo` nodes.

db2-fn:sqlquery: The function takes an SQL Select statement as its argument and returns

a sequence of XML values. The retrieved sequence is then queried with XQuery constructs. In the following example, the `INFO` column is filtered for records in the `CUSTOMER` table that have a `CID` field with a value between 1000 and 1004. Note that while SQL is not case-sensitive, XQuery is.

```
XQUERY (: Retrieve details of customers by Cid:)
declare default element namespace "http://www.altova.com/xquery/databases/db2"
;

<persons>
  { db2-fn:sqlquery( "SELECT info FROM customer WHERE CID>1000 AND CID<1004") /
    <person>
      <id>{ data( @Cid) } </id>
      <name>{ data( name) } </name>
    </person>
  }
</persons>
```

The XQuery document above returns the following output:

```
<persons xmlns="http://www.altova.com/xquery/databases/db2">
  <person>
    <id>1001</id>
    <name>Kathy Smith</name>
  </person>
  <person>
    <id>1002</id>
    <name>Jim Jones</name>
  </person>
  <person>
    <id>1003</id>
    <name>Robert Shoemaker</name>
  </person>
</persons>
```

Note the following points:

- The default element namespace declaration in the prolog applies for the entire XQuery document and is used for navigation of the XML document as well as for construction of new elements. This means that the XQuery selector `name` is expanded to `<default-element-namespace>:name`, and that constructed elements, such as `persons`, are in the default element namespace.
- The SQL Select statement is not case-sensitive.
- The `WHERE` clause of the Select statement should reference another database item—not a node inside the XML file being accessed.
- The `"/` after the `db2-fn:sqlquery` function represents the first item of the returned sequence, and this item is the context node for further navigation.

Execute the XQuery

To execute the XQuery document, select the **XQuery Execution** command (**XSL/XQuery** menu). Alternatively, press **Alt+F10** or click the XQuery Execution icon . The result of the execution is displayed in a new document.

7.3 XSLT and XQuery Debugger

Altova website:  [XSLT Debugger](#), [XQuery Debugger](#)

The XSLT and XQuery Debugger enables you to test and debug XSLT stylesheets and XQuery documents. The XSLT and XQuery Debugger interface presents simultaneous views of the XSLT/XQuery document, the result document, and the source XML document. You can then go step-by-step through the XSLT/XQuery document. The corresponding output is generated step-by-step, and, if a source XML file is displayed, the corresponding position in the XML file is highlighted for each step. At the same time, windows in the interface provide debugging information.

The XSLT and XQuery Debugger always opens within a **debugging session**. Debugging sessions can be of three types:

- XSLT 1.0, which uses the built-in Altova XSLT 1.0 engine
- XSLT 2.0, which uses the built-in Altova XSLT 2.0 engine
- XQuery, which uses the built-in Altova XQuery 1.0 engine

Which kind of debugging session is opened is determined automatically by the type of document from which the debugging session is opened (hereafter called the *active document* or *active file*). XSLT debugging sessions are opened from XSLT files (which version depends on the value of the `version` attribute of the `xsl:stylesheet` (or `xsl:transform`) element in the XSLT stylesheet ("1.0" for XSLT 1.0 and "2.0" for XSLT 2.0)). XQuery debugging sessions are opened from XQuery files. If the active file is an XML file, the selection depends on whether you choose to run an XSLT or XQuery file on the XML file; if the former, the selection further depends on whether the stylesheet is an XSLT 1.0 or XSLT 2.0 stylesheet.

This information is summarized in the table below.

Active File	Associated File	Debugging Session
XSLT 1.0	XML; (required)	XSLT 1.0 (using built-in Altova XSLT 1.0 engine)
XSLT 2.0	XML; (required)	XSLT 2.0 (using built-in Altova XSLT 2.0 engine)
XQuery	XML; (optional)	XQuery (using built-in Altova XQuery engine)
XML	XSLT 1.0, or XSLT 2.0, or XQuery; (required)	XSLT 1.0, XSLT 2.0, or XQuery. XSLT 1.0 or 2.0 depending on value of <code>version</code> attribute of <code>xsl:stylesheet</code> (or <code>xsl:transform</code>) element of XSLT stylesheet.

For details about the three Altova engines, please see the following sections:

- [Altova XSLT 1.0 Engine](#)
- [Altova XSLT 2.0 Engine](#)
- [Altova XQuery 1.0 Engine](#)

Automating XSLT and XQuery tasks with Altova XML 2012

Altova XML is a free application which contains Altova's XML Validator, XSLT 1.0, XSLT 2.0, and XQuery 1.0 engines. It can be used from the command line, via a COM interface, in Java programs, and in .NET applications to validate XML documents, transform XML documents using XSLT 1.0 and 2.0 stylesheets, and execute XQuery documents.

XSLT and XQuery tasks can therefore be automated with the use of Altova XML. For example,

you can create a batch file that calls Altova XML to transform a set of XML documents or to execute a set of XQuery documents. See the [Altova XML documentation](#) for details.

7.3.1 Mechanism and Interface

The broad mechanism used for debugging XSLT and XQuery files using the XSLT and XQuery Debugger is given below. Note that there is a difference between a debugging session and the debugger, even though both are started with the **XSL/XQuery | Start Debugger / Go** command. You must first start the debugging session, and then step through the XSLT or XQuery document with the debugger.

- Open a debugging session (with the **XSL/XQuery | Start Debugger / Go** command). The appropriate session (XSLT 1.0, XSLT 2.0, or XQuery) is selected on the basis of the active file (see [XSLT and XQuery Debugger](#)). The XSLT and XQuery Debugger works only in Text View and Grid View. If the view of the active document is not Text View or Grid View when you start the debugging session, you will be prompted for permission to change the view to Text View, which is the default view of the XSLT and XQuery Debugger. You can, in the [Debugger Settings dialog](#), also choose to have this option set permanently.
- Step through the XSLT or XQuery document using the **Step Into**, **Step Out**, and **Step Over** commands in the **XSL/XQuery** menu. (If you click the **Start Debugger / Go** command at this point, the debugger will go through the entire transformation or execution, stopping only at breakpoints. If no breakpoint has been set, it will go through the transformation in one step, without showing any debug results.) If an XML file is associated with the session, the corresponding locations in the XML file are highlighted. Simultaneously, output for corresponding steps is generated in the result file and the result document is built up step-by-step. In this way, you can analyse what each statement of the XSLT or XQuery file does.

XMLSpy Menu Bar		
Toolbar icons, including Debugger icons		
XML Document	XSLT/XQuery Document	Output File
Context (XSLT only) Variables XPath Watch	Call Stack Messages Templates (XSLT only) Info	

Alternatively to the view of the three documents (XML, XSLT/XQuery, Output) shown above, a view of two documents (XSLT/XQuery and Output), or a view of any one of the documents can be selected.

- While a debugging session is open, information windows in the interface provide information about various aspects of the transformation/execution (Variables, XPath Watch, Call Stack, Messages, Info, etc).
- While a debugging session is open, you can stop the debugger (not the same as

stopping the debugging session) to make changes to any of the documents. All the editing features that are available in your XMLSpy environment are also available while editing a file during a debugging session. When the debugger is stopped, the XSLT and XQuery Debugger interface stays open, and you have access to all the information in the information windows. After stopping the debugger in a debugging session, you can restart the debugger (from the beginning of the XSLT/XQuery document) within the same debugging session.

- Breakpoints can be set in the XSLT file to interrupt the processing at selected points. This speeds up debugging sessions since you do not have to step through each statement in the XSLT or XQuery document manually.
- Tracepoints can be set in the XSLT file. For instructions where a tracepoint is set, the value of that instruction is output when the instruction is reached.
- Stop a debugging session. This closes the XSLT and XQuery Debugger interface and returns you to your previous XMLSpy environment. The information in the information windows is no longer available. Breakpoint and tracepoint information, however, is retained in the respective files till the file is closed. (So if you start another debugging session involving a file containing breakpoints, the breakpoints will apply in the newly opened debugging session.)

Please note: The Debugger toolbar with Debugger icons appears automatically when a debugging session is started.

7.3.2 Commands and Toolbar Icons

Debugger commands are available in the **XSL/XQuery** menu and as toolbar icons. The debugger icons are automatically made available in the toolbar when a debugging session is opened. These icons are listed below.



Start Debugger/Go (Alt+F11)

Starts or continues processing the XSLT/XQuery document till the end. If breakpoints have been set, then processing will pause at that point. If the debugger session has not been started, then this button will start the session and stop at the first node to be processed. If the session is running, then the XSLT/XQuery document will be processed to the end, or until the next breakpoint is encountered. If tracepoints have been set, the value of the instruction for which the tracepoint was set is displayed in the Trace window when that instruction is reached.



View the active document only

Maximizes the window of the currently active document in the Debugger interface.



View XSLT/XQuery and Output

Displays the XSLT and Output documents in their windows, while hiding the XML document.



View XML, XSLT/XQuery and Output

Displays the XML, XSLT/XQuery, and Output documents. This is the default view when an XML document is associated for the debugging session.

**Stop Debugger**

Stops the debugger. This is not the same as stopping the debugger session in which the debugger is running. This is convenient if you wish to edit a document in the middle of a debugging session or to use alternative files within the same debugging session. After stopping the debugger, you must restart the debugger to start from the beginning of the XSLT/XQuery document.

**Step into (F11)**

Proceeds in single steps through all nodes and XPath expressions in the stylesheet. This command is also used to re-start the debugger after it has been stopped.

**Step Over (Ctrl+F11)**

Steps over the current node to the next node at the same level, or to the next node at the next higher level from that of the current node. This command is also used to re-start the debugger after it has been stopped.

**Step Out (Shift+F11)**

Steps out of the current node to the next sibling of the parent node, or to the next node at the next higher level from that of the parent node.

**Show current execution node**

Displays/selects the current execution node in the XSLT/XQuery document and the corresponding context node in the XML document. This is useful when you have clicked in other tabs which show or mark specific code in the XSLT stylesheet or XML file, and you want to return to where you were before you did this.

**Restart Debugger**

Clears the output window and restarts the debugging session with the currently selected files.

**Insert/Remove Breakpoint (F9)**

Inserts or removes a breakpoint at the current cursor position. Inline breakpoints can be defined for nodes in both the XSLT/XQuery and XML documents, and determine where the processing should pause. A dashed red line appears above the node when you set a breakpoint. Breakpoints cannot be defined on closing nodes, and breakpoints on attributes in XSLT documents will be ignored. This command is also available by right-clicking at the breakpoint location.

**Insert/Remove Tracepoint (Shift+F9)**

Inserts or removes a tracepoint at the current cursor position. Inline tracepoints can be defined for nodes in XSLT documents. During debugging, when a statement with a tracepoint is reached, the result of that statement is output in the Trace window. A dashed blue line appears above the node when you set a tracepoint. Tracepoints cannot be defined on closing nodes. This command is also available by right-clicking at the tracepoint location.

**Enable/Disable Breakpoint (CTRL+F9)**

This command (no toolbar icon exists) enables or disables already defined breakpoints. The red breakpoint highlight turns to gray when a breakpoint is disabled. The debugger does not stop at disabled breakpoints. To disable/enable a breakpoint, place the cursor in that node name and click the Enable/Disable Breakpoint command. This command is also available by right-clicking at the breakpoint location.

**Enable/Disable Tracepoint (Shift+CTRL+F9)**

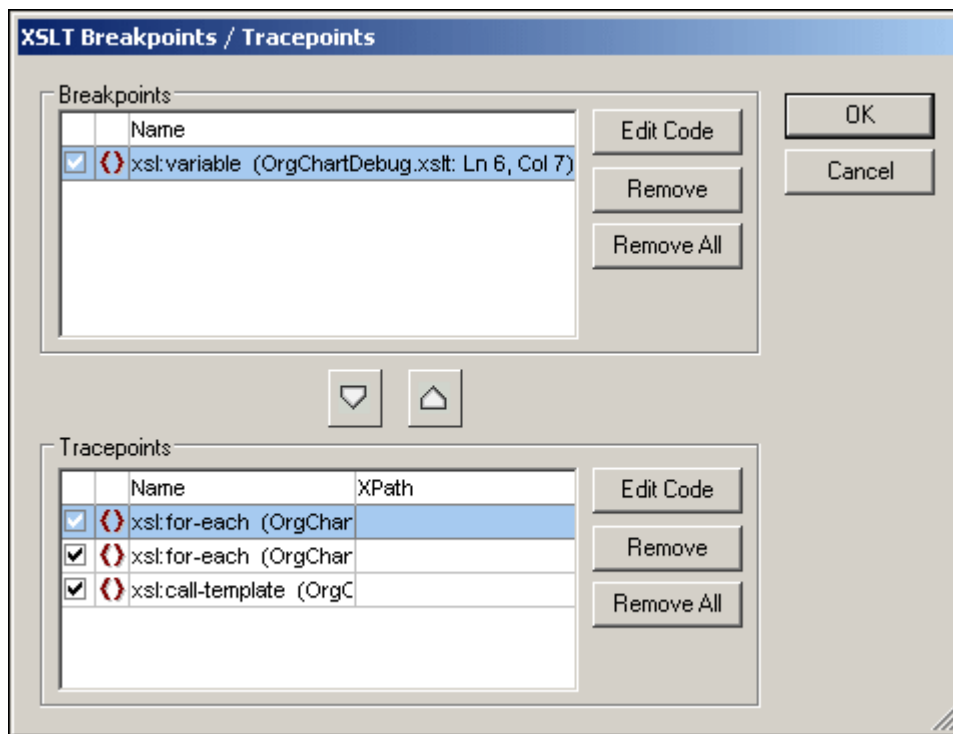
This command (no toolbar icon exists) enables or disables already defined tracepoints. The blue tracepoint highlight turns to gray when a tracepoint is disabled. No output is made in the Trace window for disabled tracepoints. To disable/enable a tracepoint, place the cursor in that node name and click the Enable/Disable Tracepoint command. This command is also available by right-clicking at the tracepoint location.

**End Debugger Session**

Ends the debugging session and returns you to the normal XMLSpy view that was active before you started the debugging session. Whether the output documents that were opened for the debugging session stay open depends on a setting you make in the [XSLT/XQuery Debugger Settings](#) dialog.

**XSLT Breakpoints / Tracepoints Dialog**

This command opens the XSLT/XQuery Breakpoints / Tracepoints dialog, which displays a list of all currently defined breakpoints/tracepoints (including disabled ones) in all files in the current debugging session.

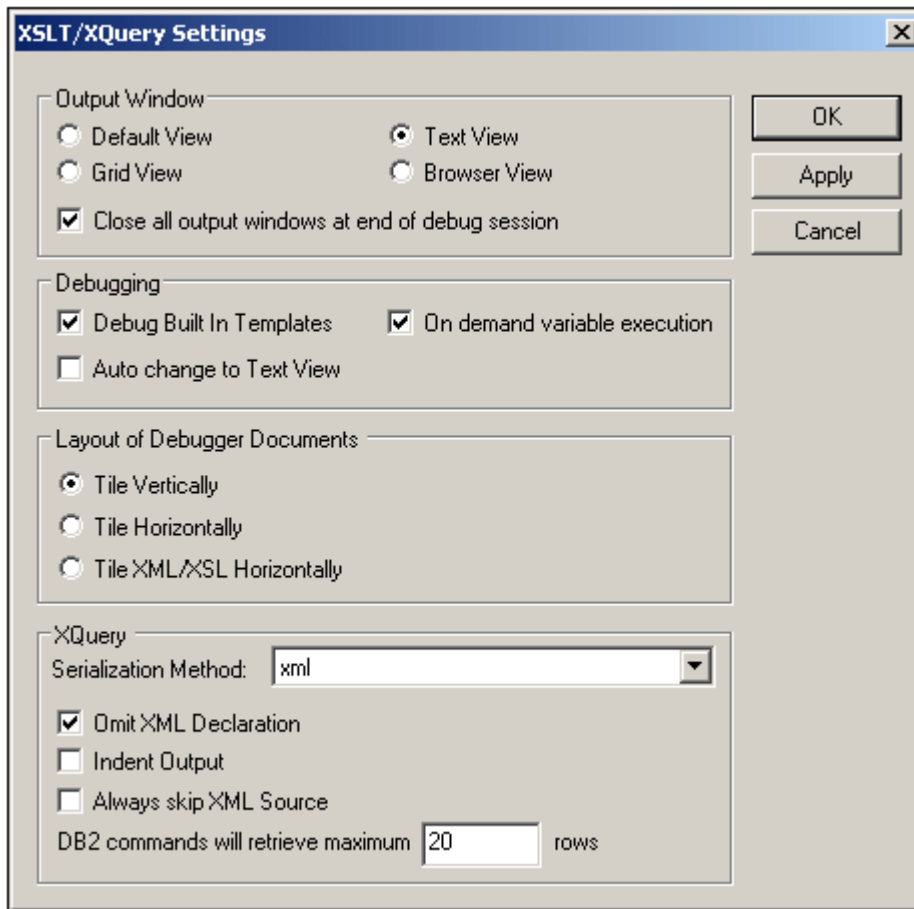


The check boxes indicate whether a breakpoint/tracepoint is enabled (checked) or disabled. You can remove the highlighted breakpoint/tracepoint by clicking the corresponding **Remove** button and remove all breakpoints or tracepoints by clicking the corresponding **Remove All** button. The corresponding **Edit Code** button takes you directly to the selected breakpoint/tracepoint in the file.

7.3.3 XSLT/XQuery Settings

The XSLT and XQuery Debugger Settings dialog enables you to set debugging and output options that are applicable to all debugging sessions. To access the Settings dialog, click

XSL/XQuery | XSLT/XQuery Settings or click the  icon in the toolbar. The different settings are described below.



Output Window

Sets the view of the output document window (Default, Text, Grid, or Browser). The Default View is that selected for the output file type in the File tab of the Options dialog ([Tools | Options](#)). For XSLT transformations, the output file type is defined in the XSLT file. For XQuery executions, the output file type is determined by the serialization format you choose in the XQuery setting of this dialog (*see below*).

The Close All Output Windows option gives you the opportunity to keep open the output document windows that were opened in the debugging session when the debugging session ends.

Debugging

The Debug Built-in Templates setting causes the debugger to **step into** built-in templates code whenever appropriate. It is not related to the **display** of built-in templates when clicking this type of template entry in the Templates tab, or if the callstack shows a node from the built-in template file.

The XSLT Debugger works only in Text View or Grid View. The Auto Change to Text View option enables you to automatically switch to the Text View of a document for debugging if a document is not in Text View or Grid View. (The XQuery Debugger works in Text View only.) If the *On demand variable execution* check box is checked, the definition of a variable will be stepped into when the variable is called. Otherwise, the Debugger will not step into the variable definition when it encounters a call to a variable, but will carry on to the next step.

Layout of Debugger Documents

The Debugger Documents are the documents that are open in the Debugger. You can select whether these documents should be tiled vertically, horizontally, or XML/XSLT horizontally with the result document tiled vertically relative to the XML and XSLT.

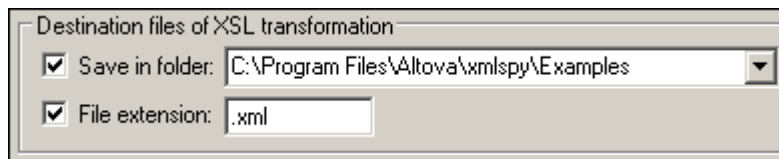
XQuery

The serialization method determines two things: (i) the file format of the generated output file, and (ii) the rules followed in writing the output to the output file. The available options are HTML, Text, XHTML, and XML. The selection you make sets up an empty file of the selected file type when you start the debugger inside a debugging session. The file type is significant because it enables currently active XMLSpy features for that file type (such as validation for XML files).

You can choose to omit the XML declaration and to indent the output. The Always Skip XML Source option enables you to always skip the optional XML file association step when you start a debugging session from an XQuery file. For DB2 databases, you can specify the maximum number of rows to be retrieved.

Alternative output setting

In the Project Properties dialog, you can make XSLT transformation associations for a folder. One of these options is the destination folder of the XSLT transformation, for which you can select a file extension.



The file type selected here determines the file type of the output format for XML or XSLT files that belong to a project for which XSLT transformation properties have been defined.

7.3.4 Starting a Debugging Session

The simplest way to start a debugging session is to start one from an XSLT, XQuery, or XML file. If the required associated file (see [Table of associated files](#)) has already been assigned to the active file, then the debugging session is started immediately. Otherwise you are prompted to select the required associated file. Since XQuery files neither require nor contain an XML file association, you can choose to be prompted for an optional XML file association each time you start an XQuery debugging session from an XQuery file, or to not be prompted.

Predefined associations

Predefined associations are relevant only for XSLT debugging sessions, and refer to cases in which the associated file assignment is already present in the active file. To make an assignment in an XML or XSLT file, do the following:

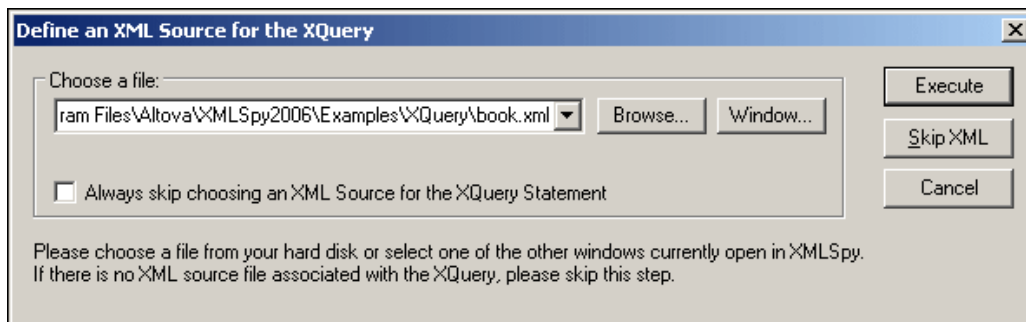
- In XML files: Open the file, click **XSL/XQuery | Assign XSL**, and select the XSLT file.
- In XSLT files: Open the file, click **XSL/XQuery | Assign sample XML file...**, and select the XML file.

When you click **XSL/XQuery | Start Debugger/Go**, the debugging session is started directly, i.e. without you being prompted for any file to associate with the active file.

Direct assignment

If no predefined association is present in the active file, you are prompted for an association. When you select **XSL/XQuery | Start Debugger/Go**, the following happens:

- For XML files: You are prompted to select an XSLT or XQuery file.
- For XSLT files: You are prompted to select an XML file.
- For XQuery files: You are given the option of selecting an XML file, which you can skip.



(The dialog shown in the screenshot appears when you start a debugging session from an XQuery file.)

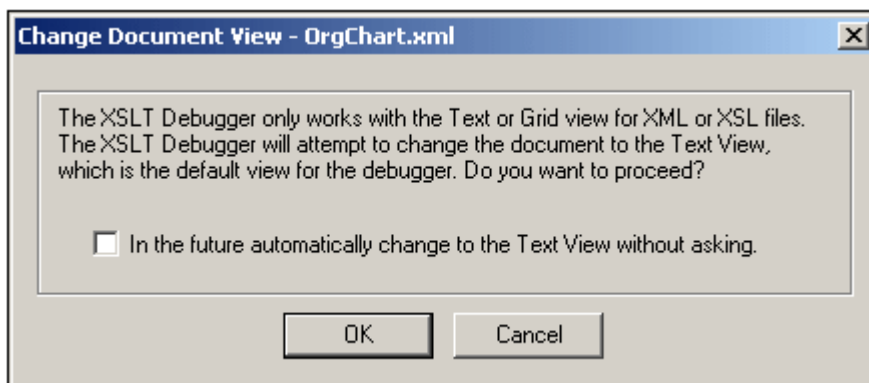
After you select the required associated file or skip an optional association, the debugging session is started.

Alternative method of file association

In the Project Properties dialog, you can make predefined associations. Click **Project | Project properties**, and assign the required files by clicking the **Use this XSL / Use this XML** check box.

Debugger View

The XSLT and XQuery Debugger works only in Text View and Enhanced Grid View. If either your XML or XSLT file is open in some other view than Text or Grid View, or if an SPS file is associated with an XML file, the following dialog pops up when you start a debugging session involving one of these files.

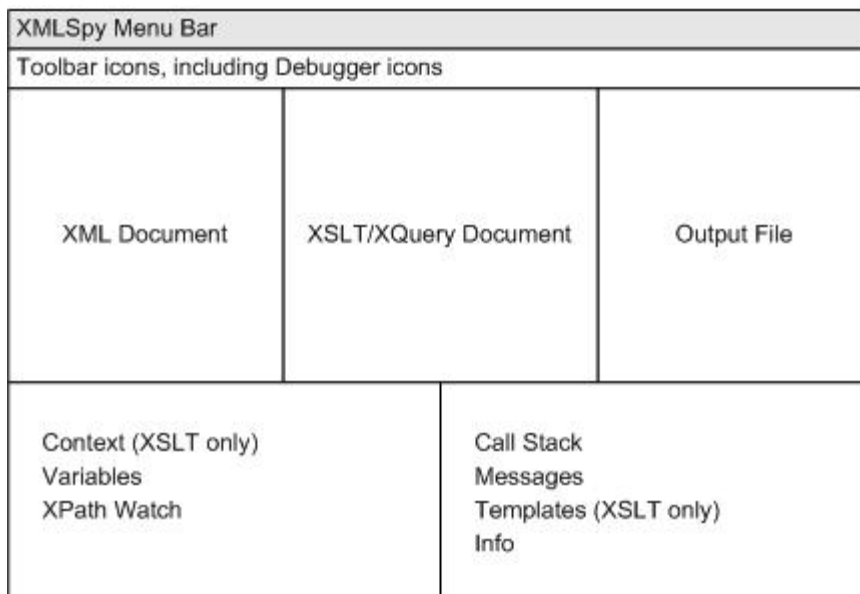


Clicking **OK** causes the document to open in Text View. Note that XQuery files are always displayed in Text View.

7.3.5 Information Windows

Information windows that are opened in the XSLT and XQuery Debugger interface during a debugging session contain information about various aspects of the XSLT transformation or XQuery execution. This information is important in helping you debug your XSLT and XQuery files.

There are eight information windows in XSLT debugging sessions and five information windows in XQuery debugging sessions. These windows are organized into two groups by default, which are located at the bottom of the XSLT and XQuery Debugger interface (*see illustration below*). These windows and the information they display are described in detail in this section.



The default layout of the XSLT and XQuery Debugger interface.

The first group of information windows displays the following windows as tabs in a single window:

- [Context](#) (for XSLT debugging sessions only)
- [Variables](#)
- [XPath-Watch](#)

The second group of information windows displays the following windows as tabs in a single window

- [Call Stack](#)
- [Messages](#)
- [Templates](#) (for XSLT debugging sessions only)
- [Info](#)
- [Trace](#)

In the default layout, therefore, there are two window groups, each having tabs for the different windows in them. One tab is active at a time. So, for example, to display information about Variables in the first information window group, click the **Variables** tab. This causes the Variables information window to be displayed and the Context and XPath-Watch information windows to be hidden. Note that in some tabs, you can use the information display as navigation tools: clicking an item can take you to that item in the XML, XSLT, or XQuery file. See the documentation of the respective information windows ([Context](#), [Call Stack](#), [Templates](#))

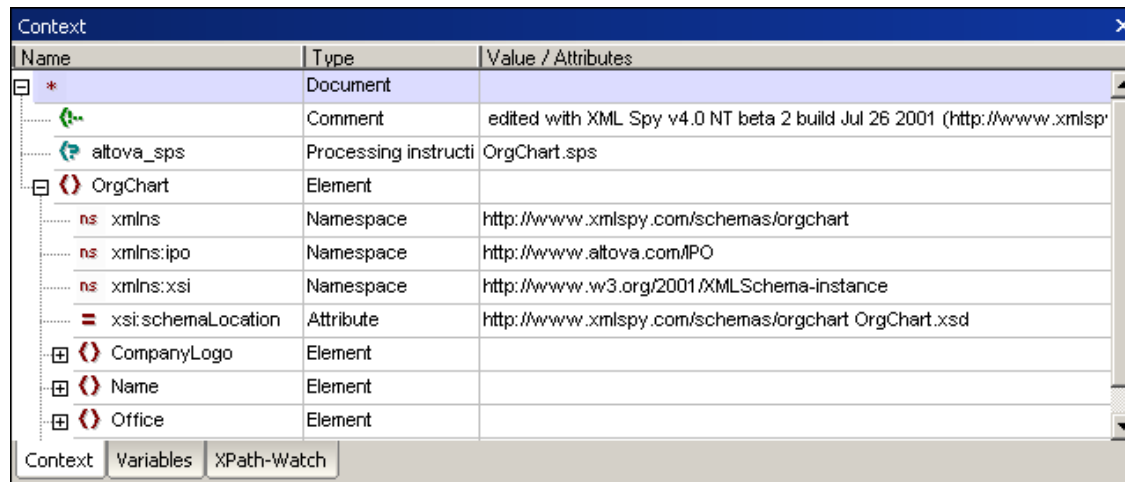
for details.

The two information window groups can be resized by dragging their borders. Individual windows can be dragged out of the containing group by clicking the tab name and dragging the window out of the group. A window can be added to a group by dragging its title bar onto the title bar of the group. Note that there is no reset button to return the layout to the default layout.

Context Window

The Context Window is available in XSLT debugging sessions only; it is not available in XQuery debugging sessions.

During the processing of the XSLT stylesheet, the processor's context is always within a template that matches some sequence (of nodes or atomic values). The Context Window displays the current processing context, which could be a sequence of nodes, a single node, or an atomic value (such as a string). Depending on the kind of a context item, its value or attribute/s is/are displayed. For example, if the context item is an element, the element's attributes are displayed. If the context item is an attribute or text node, the node's value is displayed.

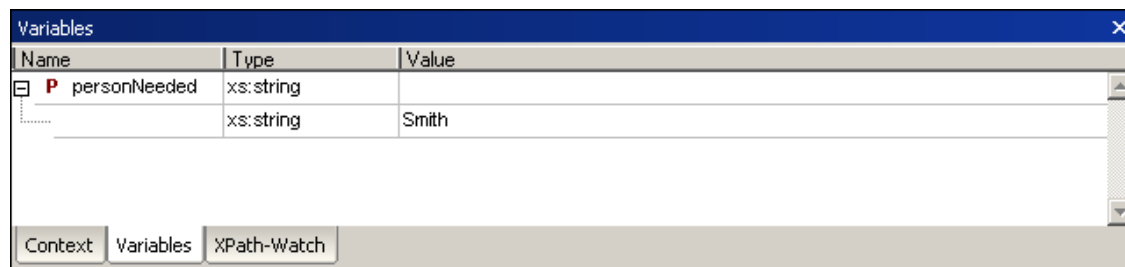


Name	Type	Value / Attributes
*	Document	
comment	Comment	edited with XML Spy v4.0 NT beta 2 build Jul 26 2001 (http://www.xmlspy.com)
altova_sps	Processing instruction	OrgChart.sps
OrgChart	Element	
xmlns	Namespace	http://www.xmlspy.com/schemas/orgchart
xmlns:ipo	Namespace	http://www.altova.com/IPO
xmlns:xsi	Namespace	http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation	Attribute	http://www.xmlspy.com/schemas/orgchart OrgChart.xsd
CompanyLogo	Element	
Name	Element	
Office	Element	

Clicking an entry in the Context Window, displays that item in the XML document. If the XML document is not currently displayed in the interface, a window for the XML document will be opened.

Variables Window

The Variables Window is available in XSLT and XQuery debugging sessions. It displays the variables and parameters that are used in the XSLT/XQuery document when they are in scope, and their values.



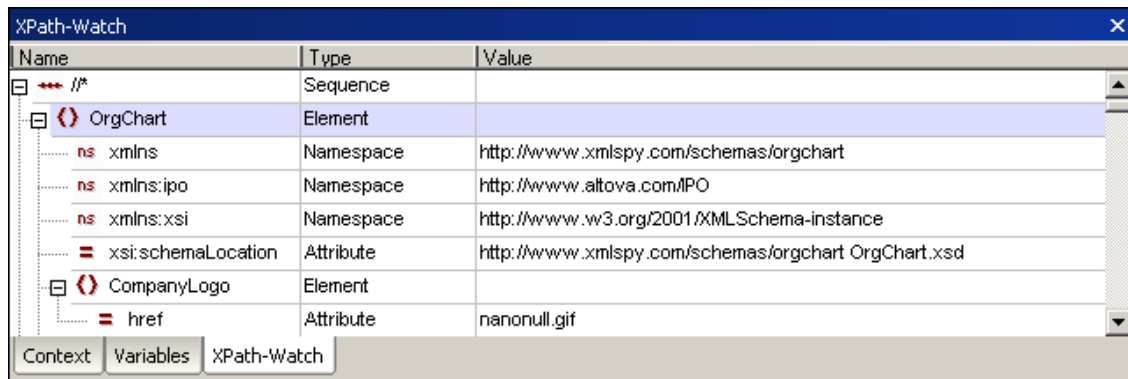
Name	Type	Value
personNeeded	xs:string	
	xs:string	Smith

Parameters are indicated with **P**, global variables (declared at top-level of a stylesheet) are indicated with **G**, and local variables (declared within an XSLT template) are indicated with **L**. The type of the values of variables and parameters is also indicated by icons in the Value field. The following types are distinguished: Node Set, Node Fragment, String, Number, and Boolean.

XPath-Watch Window

The XPath-Watch Window is available in XSLT and XQuery debugging sessions.

It enables you to enter XPath expressions that you wish to evaluate in one or more contexts. As you step through the XSLT document, the XPath expression is evaluated in the current context and the result is displayed in the Value column.



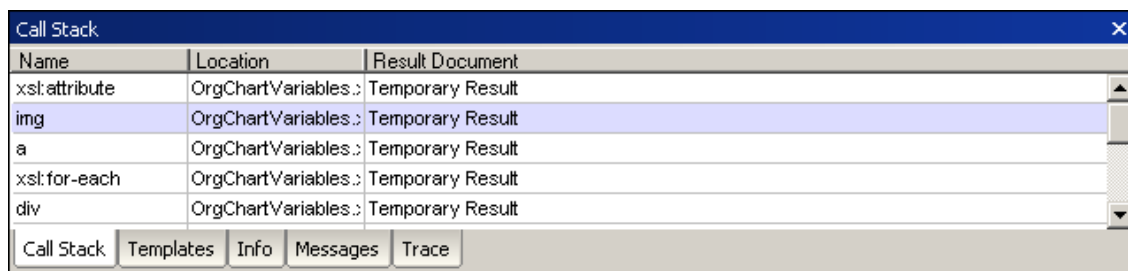
To enter an XPath expression, double-click in the text field under the Name column and enter the XPath. Alternatively, drag an XPath expression from a file and drop it into the XPath-Watch Window. Use expressions that are correct according to the XPath version that corresponds to the XSLT version of the XSLT stylesheet (XPath 1.0 for XSLT 1.0, and XPath 2.0 for XSLT 2.0).

Please note: If namespaces have been used in the XML file or XSLT file, you must use the correct namespace prefixes in your XPath expressions.

Call Stack Window

The Call Stack Window is displayed in XSLT and XQuery debugging sessions.

The Call Stack Window displays a list of previously processed XSLT templates and instructions, with the current template/instruction appearing at the top of the list.



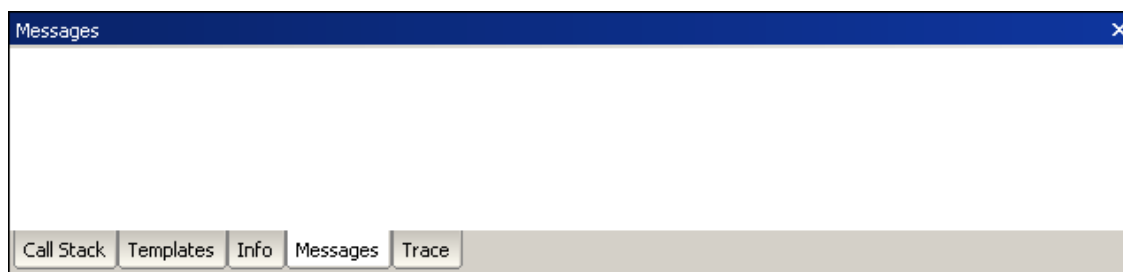
Clicking an entry in this window, causes the selected XSLT template/instruction to be displayed in the XSLT document window. Clicking a template/instruction that references a built-in template highlights the built-in template in a separate window that displays all built-in templates.

Messages Window

The Messages Window is displayed in XSLT and XQuery debugging sessions.

XSLT 1.0 and XSLT 2.0

In XSLT debugging sessions, the Messages tab displays error messages, the `xsl:message` instruction(s), or any error messages that may occur during debugging.



XQuery

In XQuery debugging sessions, the Messages Window displays error messages.

Templates Window

The Templates Window (see *screenshot*) is available in XSLT debugging sessions only; it is not available in XQuery debugging sessions.

The Templates Window displays the various templates used in the XSLT stylesheet, including built-in templates and named templates. Matched templates are listed by the nodes they match. Named templates are listed by their name. For both types of template, the mode, priority, and location of the template are displayed.

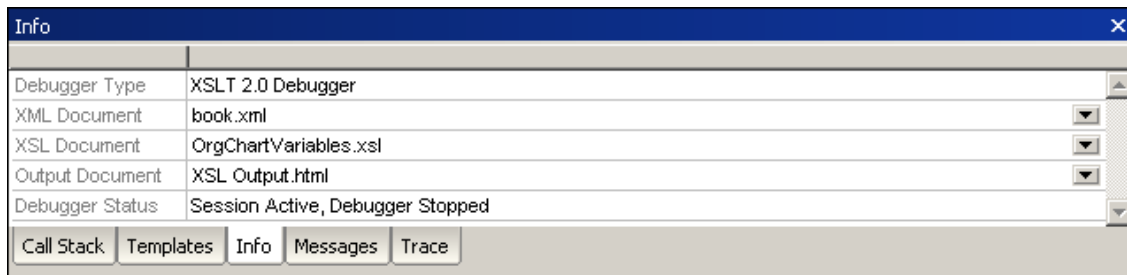
Match	Mode	Name	Priority	Location
n1:italic			0	OrgChart.xsl
n1:bold			0	OrgChart.xsl
/			-0.5	OrgChart.xsl
processing-instruction()	#all		0	Built In Templates
processing-instruction()			0	Built In Templates

In the screenshot above, there are three matched templates in the XSLT stylesheet: a template which matches the document node `/`, and templates that match the `n1:italic` and `n1:bold` nodes. All the other templates are built-in templates (indicated with no entry in the Location field).

Clicking an entry in this window, causes the template to be highlighted in the XSLT document window. If you click a built-in template, the template is highlighted in a separate window that displays all the built-in templates.

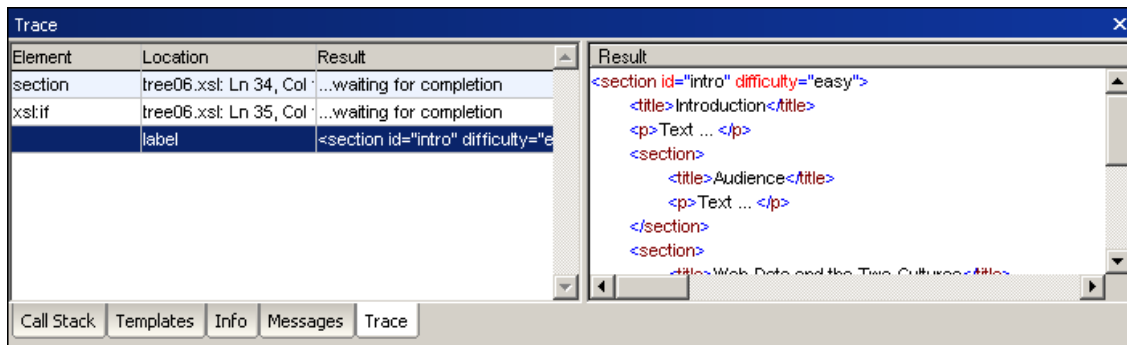
Info Window

The Info Window is available in XSLT and XQuery debugging sessions. It provides meta information about the current debugging session. This information includes what debugger is being used, the names of the source and output documents, and the status of the debugger.



Trace Window

The Trace Window is displayed in XSLT and XQuery debugging sessions.



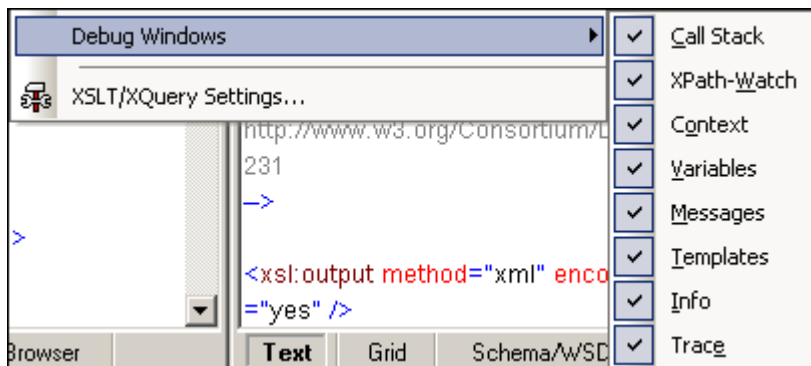
The Trace Window contains the element the tracepoint is set for, its location in the XSLT stylesheet and the result generated when that element is executed. Click on a row in the left side of the window to display the full result on the right.

Arranging the Info Windows

The Information Windows can be arranged inside the XSLT and XQuery Debugger interface. Windows can be docked in the interface, can float in it, and can be arranged as a collection of panes in a window. You can use the following mechanisms to arrange the windows.

Menu

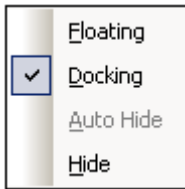
In the **XSL/XQuery** menu, placing the cursor over the item **Debug Windows** pops up the list of Info Windows. You can hide or show individual windows by clicking the window.



This toggles the display of the window on and off.

Context Menu

The context menu can be accessed by right-clicking a window tab or title bar.



Click the required option to cause that window to float, be docked or be hidden.

Drag-and-drop

You can drag a window by its tab or title bar and place it at a desired location.

Additionally, you can dock the window in another window or in the interface using placement controls that appear when you drag a window:

- When you drag a window over another window, a circular placement control appears (see *screenshot below*). This control is divided into five placement sectors. Releasing the mouse key on any of these sectors docks the dragged window into the respective sector of the **target window**. The four arrow sectors dock the dragged window into the respective sides of the target window. The center button docks the dragged window as a tab of the target window. You can also dock a window as a tab in another window by dragging it to the tab bar and dropping it there.



- When you drag a window, a placement control consisting of four arrows appears. Each arrow corresponds to one side of the **Debugger interface**. Releasing a dragged window over one of these arrows docks the dragged window into one side of the Debugger interface.



You can also double-click the title bar of a window to toggle it between its docked and floating positions.

7.3.6 Breakpoints

The XSLT and XQuery Debugger enables you to define breakpoints in XSLT, XQuery, and XML documents. Breakpoints are displayed as a dashed red line (*shown in the screenshot below*).

Please note: It is possible to set a tracepoint and a breakpoint for the same instruction. This appears as a dashed blue and red line (*see screenshot*).

```
<!-- Local Variable reused -->
<xsl:variable name="OfficeName" select="n1:Name"/>
<!-- Display the company name if the variable is true ***** -->
<xsl:if test="$Show_Company_Name">
  <h3>
    <xsl:value-of select="$OfficeName"/>
  </h3>
  <xsl:message>
    <xsl:text>Company Named Displayed</xsl:text>
  </xsl:message>
</xsl:if>
```

When you start the debugger within a debugging session, the debugging will pause at each encountered breakpoint. In this way, you can identify specific areas to debug, and restrict attention to these areas in either the XSLT, XQuery, and/or XML documents. You can set any number of breakpoints.

Please note: Breakpoints set for a document remain in that document until it is closed. However, if you switch to Schema View (for example, in the case of XSD documents), then the breakpoints are deleted; when you switch back to Text View or Grid View (from Schema View), there will be no breakpoint.

Breakpoints in XML documents

You can set breakpoints on any node in an XML document. The break in processing will occur at the start of that node.

Breakpoints in XSLT documents

You can set breakpoints at the following points in an XSLT document:

- At the beginning of templates and template instructions (e.g., `xsl:for-each`).
- On an XPath expression (XPath 1.0 or XPath 2.0).
- On any node in a literally constructed XML fragment. The break in processing will occur at the start of that node.

Breakpoints in XQuery documents

You can set breakpoints at the following points in an XQuery document:

- At the beginning of XQuery statements.
- In an XQuery expression.
- On any node in a literally constructed XML fragment. The break in processing will occur at the start of that node.

Inserting/removing breakpoints

To insert a breakpoint:

1. Place the cursor at the point in the document where you wish to insert the breakpoint (see *paragraphs above*). In XSLT debugging sessions, you can set breakpoints in both Text View and Grid View. XQuery debugging sessions are available only in Text View.
2. Do one of the following:
 - Select **XSL/XQuery | Insert/Remove Breakpoint**.
 - Press **F9**.

- Right-click and select **Insert/Remove Breakpoint**.

To remove a breakpoint:

1. Place the cursor at the point in the document containing the breakpoint.
2. Do one of the following:
 - Select **XSL/XQuery | Insert/Remove Breakpoint**.
 - Press **F9**.
 - Right-click and select **Insert/Remove Breakpoint**.

Alternatively, you can use the Breakpoints dialog to remove a breakpoint:

1. Select the menu option **XSL/XQuery | Breakpoints...**
2. Click the breakpoint in the dialog box and click **Remove**.

The **Remove All** button deletes all the breakpoints from the dialog box (and all XSLT stylesheets).

Disabling/enabling breakpoints:

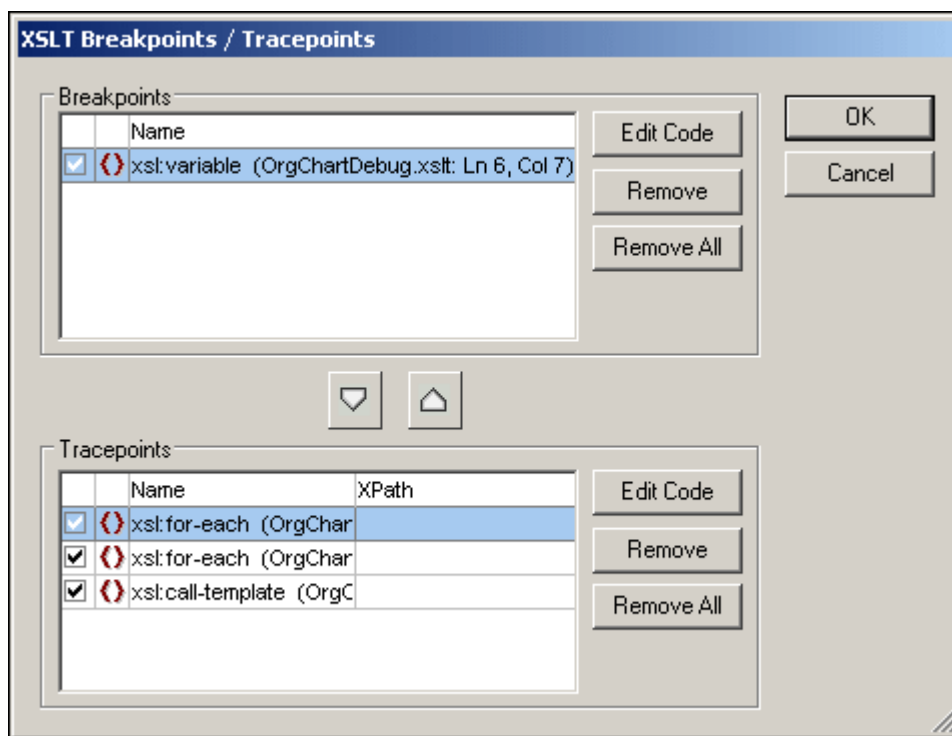
After inserting breakpoints, you can disable them if you wish to skip over breakpoints without having to delete them. You can enable them again when necessary.

To disable a breakpoint:

1. Place the cursor in the node or expression containing the breakpoint.
2. Select **XSL/XQuery | Enable/Disable Breakpoint** (or press **Ctrl+F9**). The breakpoint changes from red to gray, indicating that it has been disabled.

Alternatively, you can use the Breakpoints dialog to disable a breakpoint:

1. Select the menu option **XSL/XQuery | Breakpoints/Tracepoint...**. This opens the XSLT Breakpoints / Tracepoints dialog box which displays the currently defined breakpoints in all open XML source and XSLT stylesheet documents.



2. Remove the check mark of the breakpoints you wish to disable, and click **OK** to confirm. The breakpoint changes from red to gray, indicating that it has been disabled.

To enable a breakpoint:

1. Place the cursor in the node or expression containing the breakpoint.
2. Select **XSL/XQuery | Enable/Disable Breakpoint** (or press **Ctrl+F9**). The breakpoint changes from gray to red, indicating that it has been enabled.

Finding a specific breakpoint

To find a specific breakpoint:

1. Select the menu option **XSL/XQuery | Breakpoints/Tracepoints....**. The XSLT Breakpoints / Tracepoints dialog appears.
2. Click the required breakpoint in the breakpoint list.
3. Click the **Edit Code** button. The Breakpoints dialog box is closed and the text cursor is placed directly in front of the breakpoint in Text view. In the Enhanced Grid view, the table cell containing the breakpoint is highlighted in red.

Continuing debugging after a breakpoint

To continue debugging after a breakpoint:

- Select the **XSL/XQuery | Step into** or **XSL/XQuery | Start Debugger/Go** command.

7.3.7 Tracepoints

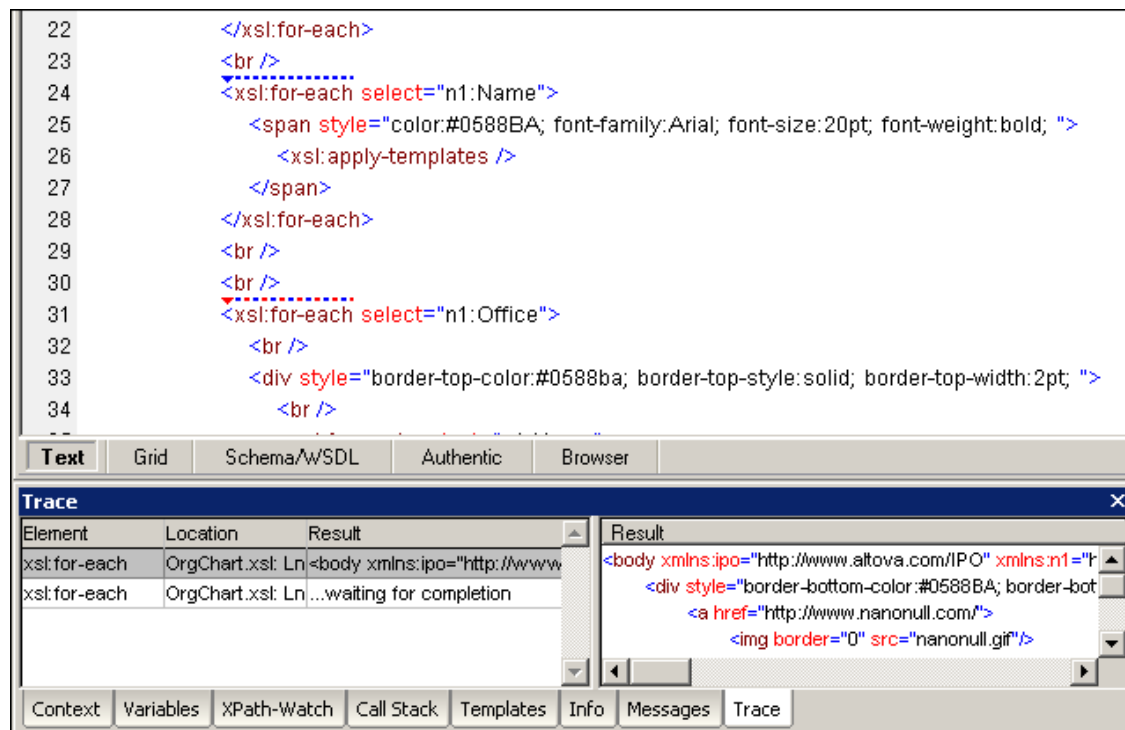
The XSLT and XQuery Debugger enables you to define tracepoints in XSLT documents.

Tracepoints allow you to trace content generated by an instruction or view the result of an XPath

expression at the point where the tracepoint is set without having to edit the XSLT stylesheet, for example, using the `xsl:message` element to output debugging messages.

Tracepoints are displayed as a dashed blue line in XSLT stylesheets (*shown in the screenshot below*).

Please note: It is possible to set a tracepoint and a breakpoint for the same instruction. This appears as a dashed blue and red line (*see screenshot*).



The debugger outputs the content generated by each instruction that has a tracepoint set for it. This output is visible in the Trace window. You can set any number of tracepoints in an XSLT stylesheet.

Please note: Tracepoints set for a document remain in that document until it is closed.

Tracepoints in XSLT documents

You can set tracepoints on XSL instructions and literal results in an XSLT stylesheet.

Tracepoints in XML and XQuery documents

You can set tracepoints in XML and XQuery documents, however, these tracepoints have no effect.

Inserting/removing tracepoints

To insert a tracepoint:

1. Place the cursor at the point in the XSLT document where you wish to insert the tracepoint. During debugging sessions, you can set tracepoints in both Text View and Grid View.
2. Do one of the following:

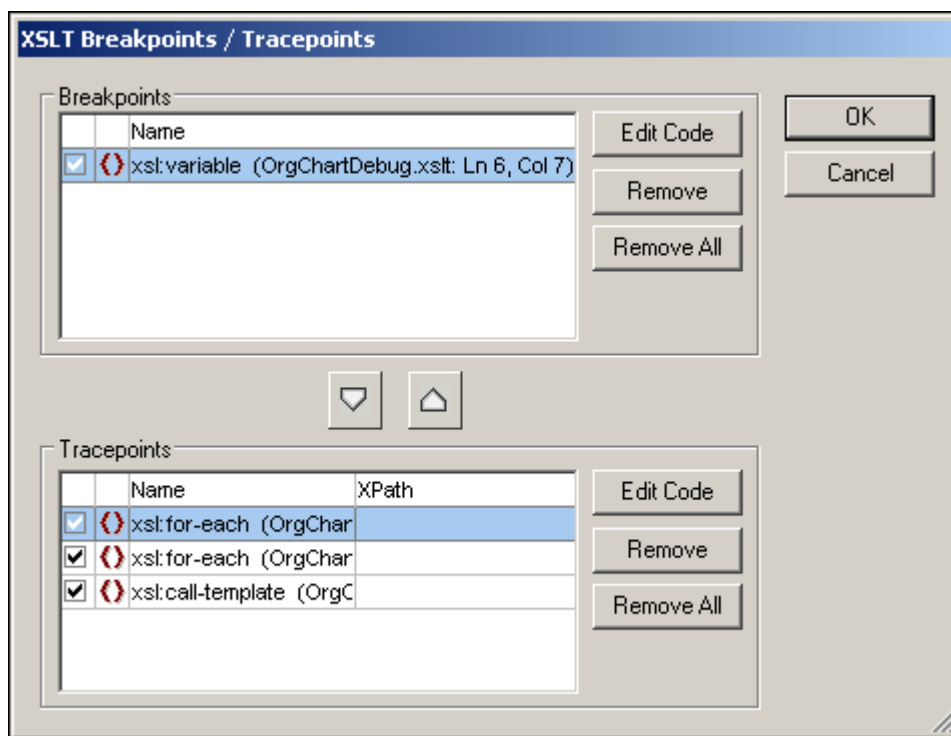
- Select **XSL/XQuery | Insert/Remove Tracepoint**.
- Press **Shift+F9**.
- Right-click and select **Insert/Remove Tracepoint**.

To remove a tracepoint:

1. Place the cursor at the point in the XSLT document containing the tracepoint.
2. Do one of the following:
 - Select **XSL/XQuery | Insert/Remove Tracepoint**.
 - Press **Shift+F9**.
 - Right-click and select **Insert/Remove Tracepoint**.

Alternatively, you can use the XSLT Breakpoints / Tracepoints dialog to remove a tracepoint:

1. Select the menu option **XSL/XQuery | Breakpoints/Tracepoints....**
2. Click the tracepoint in the dialog box (see screenshot) and click **Remove**.



The **Remove All** button in the Tracepoints pane deletes all the tracepoints from the dialog box (and from all XSLT stylesheets).

Setting an XPath for a tracepoint

You can set an XPath for a tracepoint. When you set an XPath for a tracepoint, the result of the evaluation of the XPath is displayed in the Trace window instead of the content generated by the statement for which the tracepoint is set. The XPath is evaluated relatively to the context node at the point where the tracepoint is set.

To set an XPath for a tracepoint:

1. Select the menu option **XSL/XQuery | Breakpoints/Tracepoints....** This opens the XSLT Breakpoints / Tracepoints dialog box which displays the currently defined tracepoints in all open XSLT stylesheet documents.
2. Enter the XPath in the **XPath** column in the row that corresponds to the tracepoint.

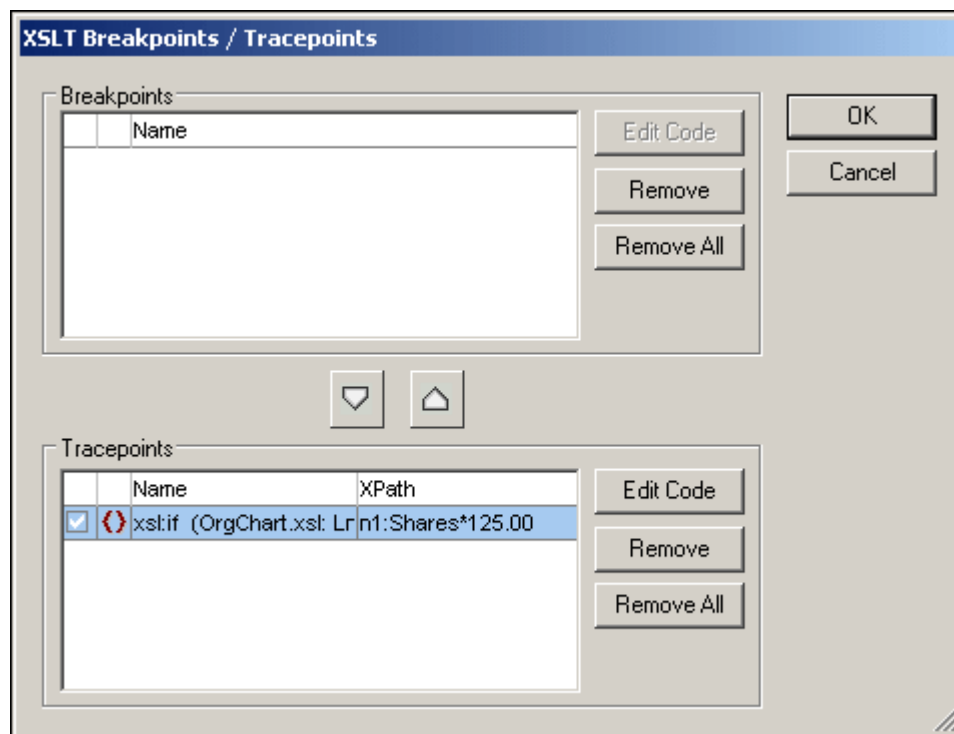
Example:

The following example uses the file `OrgChart.xml`, which is found in the XMLSpy Examples folder.

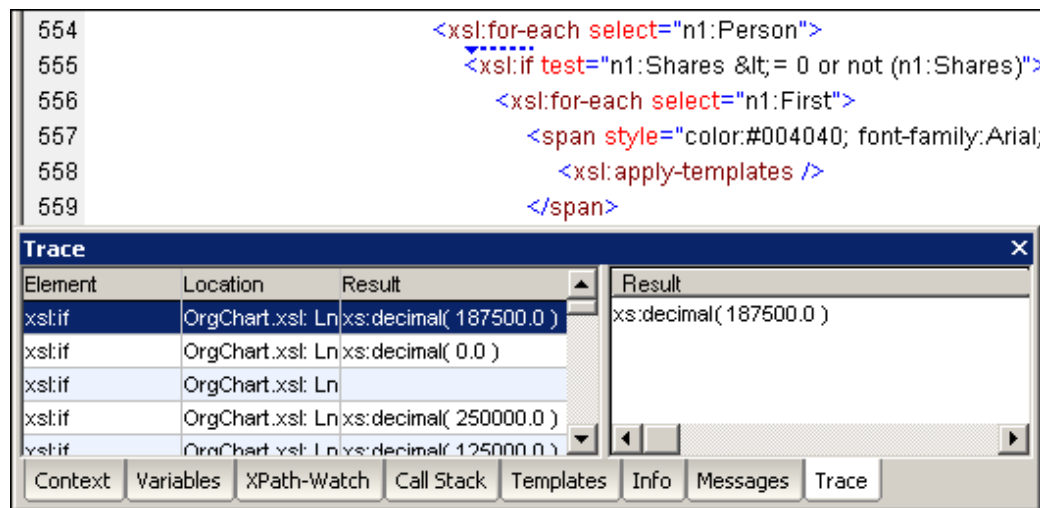
The tracepoint is set such that the context node is `Person`. The `Person` element contains a `Shares` element. We want to display the number of shares that each person has, multiplied by 125 (the value of each share).

Do the following:

1. Open the file `OrgChart.xml`.
2. Set a tracepoint at line 555.
3. Open the XSLT Breakpoints / Tracepoints dialog and enter the XPath `nl:Shares*125.00` for the tracepoint you just set.



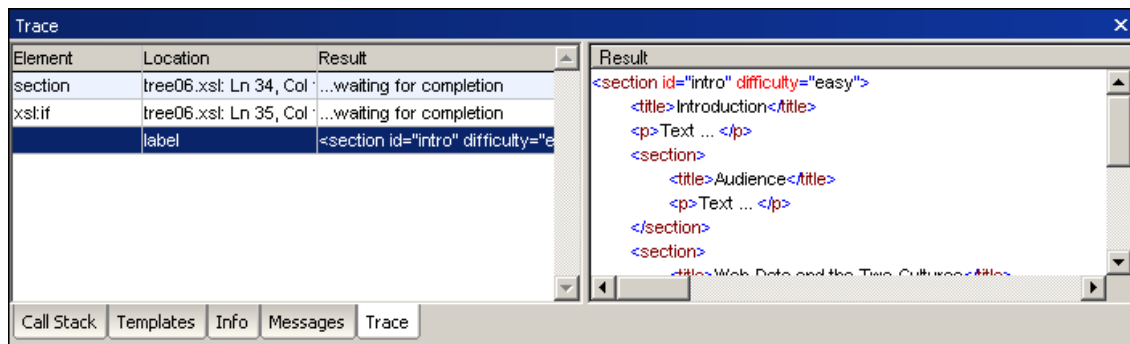
4. Start the Debugger. The results of the XPath you entered for the tracepoint appear in the Trace window.



The Trace window

Select **XSL/XQuery | Start Debugger/Go** to start debugging. The output of instructions for which tracepoints are set is displayed in the [Trace window](#) (see *screenshot*). Click a row in the Trace window to display the full result of that statement in the right side of the dialog (see *screenshot*).

Please note: Results are displayed in the Trace window only after the traced instruction is completed.



Disabling/enabling tracepoints

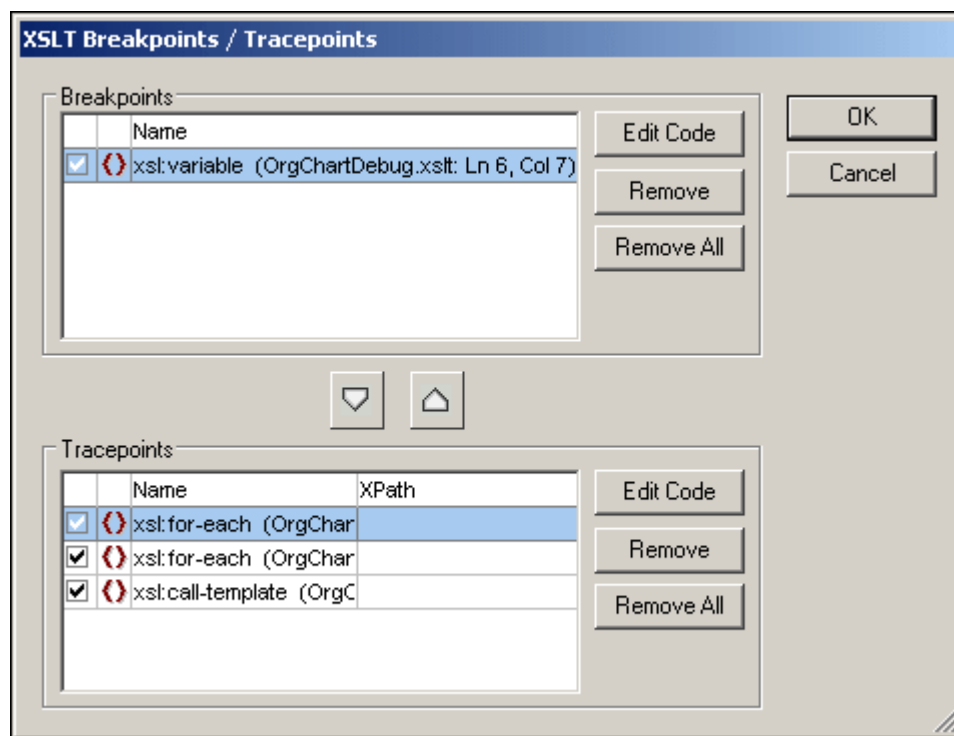
After inserting tracepoints, you can disable them if you wish to skip over them without having to delete them. You can enable them again when necessary.

To disable a tracepoint:

1. Place the cursor at the point in the XSLT stylesheet containing the tracepoint.
2. Select **XSL/XQuery | Enable/Disable Tracepoint** (or press **Ctrl+Shift+F9**). The tracepoint changes from blue to gray, indicating that it has been disabled.

Alternatively, you can use the XSLT Breakpoints / Tracepoints dialog to disable a tracepoint:

1. Select the menu option **XSL/XQuery | Breakpoints/Tracepoints....** This opens the XSLT Breakpoints / Tracepoints dialog box which displays the currently defined tracepoints in all open XSLT stylesheet documents.



2. Remove the check mark of each tracepoint you wish to disable, and click **OK** to confirm. The tracepoints change from blue to gray, indicating that they have been disabled.

To enable a tracepoint:

1. Place the cursor at the point in the XSLT document containing the tracepoint.
2. Select **XSL/XQuery | Enable/Disable Tracepoint** (or press **Ctrl+Shift+F9**). The tracepoint changes from gray to blue, indicating that it has been enabled.

Finding a specific tracepoint

To find a specific tracepoint:

1. Select the menu option **XSL/XQuery | Breakpoints/Tracepoints....**. The XSLT Breakpoints / Tracepoints dialog appears.
2. Click the required tracepoint in the tracepoint list.
3. Click the **Edit Code** button. The XSLT Breakpoints / Tracepoints dialog box is closed and the text cursor is placed directly in front of the tracepoint in Text view of the XSLT document. In Enhanced Grid view, the table cell containing the tracepoint is highlighted in blue.

7.4 XSLT and XQuery Profiler

Altova website:  [XSLT Profiler](#), [XQuery Profiler](#)

The XSLT/XQuery Profiler is a tool that is used to analyze the execution times of XSLT (1.0 and 2.0) stylesheets and XQuery documents from within XMLSpy. It tells you how much time each instruction in the XSLT stylesheet or XQuery document takes to execute, and you can use this information to optimize the execution time of these files.

The Profiler is used to find the instructions that have the highest total execution time so that this information can then be used to optimize these instructions. Instructions can have a high total execution time for one or both of the following reasons:

- the instruction is time-intensive
- the instruction is evaluated often (high hit count)

Hitcount and Callgraph Profiling

The Profiler lets you choose between *hitcount* and *callgraph* profiling. Both types of profiling show execution time statistics for each instruction.

For optimization purposes, you normally use hitcount profiling, which displays one line in the profiler for each instruction.

Callgraph profiling shows the entire execution history of an XSLT transformation or XQuery execution, i.e., which templates/functions were called, and in which order, during the transformation. In the results of callgraph profiling, there is one line for each time an instruction is called, rather than one line for each instruction.

To use the XSLT/XQuery Profiler, see [Analyzing XSLT Transformations](#) or [Analyzing XQuery Execution](#).

Profiler Views

The results of the analysis can be viewed in either of the following views by clicking the corresponding tab:

- **List View:** The profiling statistics are displayed as a list that can be sorted by, e.g., duration of instruction execution or duration of execution of the instruction and its descendants.

List								
Index	Name	Hit Count	Duration (ms)	%	Des...	Descendants an...	%	XPath
0	xsl:stylesheet	1	3.69	1.17	311.83	315.52	99.99	0.00
1	xsl:output	0	0.00	0.00	0.00	0.00	0.00	0.00
2	xsl:template	1	0.02	0.01	311.81	311.83	98.82	0.00
3	html	1	0.11	0.03	311.70	311.81	98.82	0.00
4	head	1	0.06	0.02	0.05	0.11	0.03	0.00
5	title	1	0.05	0.02	0.00	0.05	0.02	0.00
6	body	1	0.04	0.01	311.55	311.60	98.75	0.00
7	xsl:for-each	1	0.22	0.07	311.33	311.55	98.74	0.18
8	xsl:for-each	1	0.18	0.06	1.30	1.49	0.47	0.17
9	div	1	0.05	0.02	1.25	1.30	0.41	0.00
10	style	1	0.03	0.01	0.00	0.03	0.01	0.00
Tree List								

- **Tree View:** The statistics are displayed in a tree-like structure. It is possible to see, e.g., how long a function took to execute, and then expand the tree for that function and see

how much time each instruction in the function took to execute, and how many times it was executed.

Tree								
Name	Hit Count	Duration (ms)	%	Descendants (ms)	Descendants and Self...	%	XPath	
[-] xsl:stylesheet	1	3.69	1.17	311.83	315.52	99.99	0.00	
[-] xsl:output	0	0.00	0.00	0.00	0.00	0.00	0.00	
[-] xsl:template	1	0.02	0.01	311.81	311.83	98.82	0.00	
[-] html	1	0.11	0.03	311.70	311.81	98.82	0.00	
[-] head	1	0.06	0.02	0.05	0.11	0.03	0.00	
[-] body	1	0.04	0.01	311.55	311.60	98.75	0.00	
[-] xsl:template	1	0.01	0.00	0.29	0.30	0.10	0.00	
[-] span	1	0.05	0.02	0.24	0.29	0.09	0.00	
[-] style	1	0.02	0.01	0.00	0.02	0.01	0.00	
[-] xsl:apply-temp...	1	0.16	0.05	0.06	0.22	0.07	0.07	
[-] xsl:template	2	0.02	0.01	0.48	0.50	0.16	0.00	

Sorting Results

After you have run the Profiler, you can sort by the amount of time an instruction took to execute, or by the number of times that instruction was called.

To sort information in the Profiler:

1. Click the **List** tab.
2. Click the column header of the column you want to sort by (e.g., **Hit Count** to sort by the number of times an instruction was called or **Duration** to sort by the time the instruction takes to execute).

This screenshot shows the contents of the Profiler sorted by instruction duration in descending order.

List								
Index	Name	Hit Count	Duration (ms)	%	Des...	Descendants an...	%	XPath
740	xsl:value-of	7	18.33	5.81	0.00	18.33	5.81	18.02
706	xsl:value-of	7	10.46	3.31	0.00	10.46	3.31	10.16
712	xsl:value-of	7	9.60	3.04	0.00	9.60	3.04	8.90
830	xsl:apply-templates	30	7.94	2.52	2.02	9.96	3.16	3.51
734	xsl:value-of	7	7.77	2.46	0.00	7.77	2.46	7.38
549	Text	2	7.19	2.28	0.00	7.19	2.28	0.00
790	xsl:apply-templates	20	7.15	2.27	1.43	8.58	2.72	0.95
806	xsl:apply-templates	10	7.01	2.22	1.14	8.15	2.58	3.34
857	xsl:value-of	30	6.28	1.99	0.00	6.28	1.99	5.09
835	span	27	5.00	1.58	5.21	10.21	3.24	0.00
819	xsl:apply-templates	30	4.57	1.45	2.24	6.81	2.16	1.12

Optimizing Your XSLT Stylesheets and XQuery Documents

Keep in mind the following guidelines when optimizing the execution time of instructions in XSLT stylesheets and XQuery documents:

- Avoid using variables in an instruction if the variable is used only once, because initializing a variable can be time-consuming.

The following XSLT code fragments show an example of how to optimize code by removing unnecessary variables. Both do the same thing, but the second example does so without using the variables `name` and `containsResult`:

Code fragment 1:

```
<xsl:for-each select="row">
  <xsl:variable name="row" select="."/>
  <xsl:for-each select="@name">
    <xsl:variable name="name" select="."/>
    <xsl:variable name="containsResult" select="fn:contains
($name, '.exe')"/>
    <xsl:if test="string($containsResult)='true'">
      ...
    </xsl:if>
  </xsl:for-each>
</xsl:for-each>
```

The screenshot below shows the results of the analysis of the file that contains this code fragment, sorted by duration of instructions. The instruction in which the variable `containsResult` is initialized needs about 19 seconds total execution time.

Index	Name	Hit Count	Duration (ms)	%	Descendants (ms)	Descendants and Self (ms)	%	XPath
8	xsl:for-each	11238	78020.29	44.32	60755.94	138776.23	78.83	77372.08
11	xsl:if	11238	59036.65	33.54	1719.30	60755.94	34.51	58914.32
6	xsl:for-each	1	36958.65	20.99	138776.23	175734.89	99.83	45.40
10	xsl:variable	11238	19369.42	11.00	0.00	19369.42	11.00	19120.45
9	xsl:variable	11238	1768.70	1.00	0.00	1768.70	1.00	1588.65
15	xsl:for-each	262	966.90	0.55	180.34	1147.24	0.65	963.30
12	row	262	500.50	0.28	1218.80	1719.30	0.98	0.00
0	xsl:stylesheet	1	153.46	0.09	175887.34	176040.81	100.00	0.00
17	xsl:value-of	262	131.44	0.07	0.00	131.44	0.07	5.98
16	xsl:attribute	262	48.90	0.03	131.44	180.34	0.10	0.00
7	xsl:variable	262	48.73	0.03	0.00	48.73	0.03	43.36

The screenshot below shows the results in the tree view. Here we can see that the if-statement that uses the variable `containsResult` needs about 50 seconds total execution time:

The screenshot shows the XSLT and XQuery Profiler interface. The top pane displays XSLT code with line numbers 7 through 18. The code includes nested `<xsl:for-each>`, `<xsl:variable>`, and `<xsl:if>` elements. The bottom pane shows a performance tree with columns: Name, Hit Count, Duration (ms), %, Descendants (ms), Descendants and Self (ms), %, and XPath. The tree is expanded to show the `row` element, which has a hit count of 262 and a duration of 199.53 ms.

Name	Hit Count	Duration (ms)	%	Descendants (ms)	Descendants and Self (ms)	%	XPath
xsl:stylesheet	1	11.00	0.01	76949.70	76960.70	100.00	0.00
xsl:output	0	0.00	0.00	0.00	0.00	0.00	0.00
xsl:template	1	0.01	0.00	76911.73	76911.74	99.94	0.00
root	1	82.97	0.11	76828.76	76911.73	99.94	0.00
xsl:attribute	1	54.04	0.07	0.02	54.06	0.07	0.00
xsl:for-each	1	2614.55	3.40	74160.15	76774.70	99.76	43.30
xsl:variable	262	44.36	0.06	0.00	44.36	0.06	39.17
xsl:for-each	11238	22196.25	28.84	51963.90	74160.15	96.36	21573.19
xsl:variable	11238	1619.42	2.10	0.00	1619.42	2.10	1448.78
xsl:variable	11238	13286.12	17.26	0.00	13286.12	17.26	13083.39
xsl:if	11238	50441.78	65.54	1522.12	51963.90	67.52	50330.88
row	262	199.53	0.26	1322.59	1522.12	1.98	0.00

The XSLT transformation takes a total of about 74 seconds:

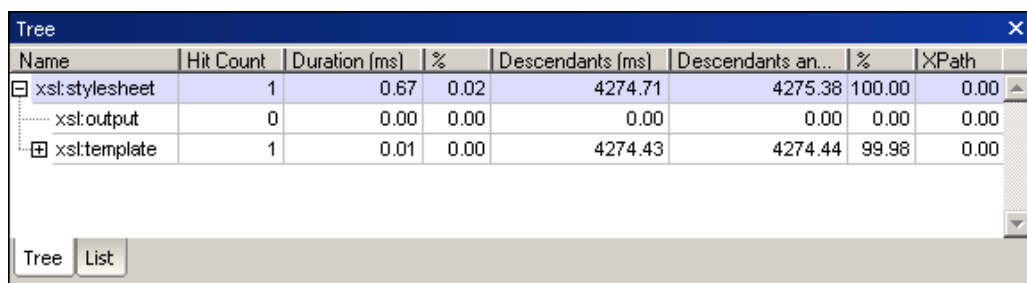
The screenshot shows the XSLT and XQuery Profiler interface with a performance tree. The tree is expanded to show the `row` element, which has a hit count of 262 and a duration of 199.53 ms.

Name	Hit Count	Duration (ms)	%	Descendants (ms)	Descendants and Self (ms)	%	XPath
xsl:stylesheet	1	20.33	0.03	73882.64	73902.97	100.00	0.00
xsl:output	0	0.00	0.00	0.00	0.00	0.00	0.00
xsl:template	1	0.01	0.00	73856.08	73856.09	99.94	0.00

Code fragment 2:

```
<xsl:for-each select="row">
  <xsl:variable name="row" select="."/>
  <xsl:for-each select="@name">
    <xsl:if test="fn:contains(., '.exe')">
      ...
    </xsl:if>
  </xsl:for-each>
</xsl:for-each>
```

After the stylesheet is rewritten without using these variables, its total execution time is only about 4.3 seconds:



Name	Hit Count	Duration (ms)	%	Descendants (ms)	Descendants an...	%	XPath
xsl:stylesheet	1	0.67	0.02	4274.71	4275.38	100.00	0.00
xsl:output	0	0.00	0.00	0.00	0.00	0.00	0.00
xsl:template	1	0.01	0.00	4274.43	4274.44	99.98	0.00

- Use variables if a value or expression is used repeatedly.
- Avoid creating local constant variables within a function; create global variables instead.
- Avoid creating constant tree fragments inside a function; create them globally instead.
- Limit your use of predicates, since filtering with predicates is evaluated separately for every node. You can reduce the number of calls to predicates, for example, by prefiltering using names. In this example, * is used with two predicates:

```
//*[ node-name( ) =Book][ author="Steve"]
```

In this equivalent statement, the name `Book` and only one predicate are used:

```
//Book[ @Author="Steve"]
```

- Split up instructions such that parts of the instruction that only need to be executed once are only executed once. Create global variables from parts that are only dependent on the global context.

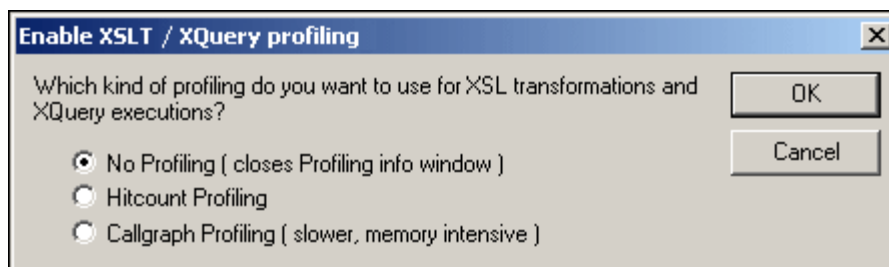
7.4.1 XSLT Profiling

Starting the Profiler

Please note: Execution time results displayed in the Profiler may be influenced by other applications that are running on your computer. When analyzing files using the Profiler, it is best to run only the XMLSpy application.

To analyze an XSLT stylesheet:

1. In XMLSpy, open the XML file that will be used as input data for the XSLT transformation.
2. Activate the Profiler by selecting **XSL/XQuery | Enable XSLT / XQuery profiling**. A dialog opens.



3. Select **Hitcount Profiling** or **Callgraph Profiling**. Click **OK** to confirm. An empty Profiler window appears.
4. Run the XSL transformation (**XSL/XQuery | XSL Transformation**). A dialog opens in which you select the path to the XSLT stylesheet you want to analyze. When the transformation is finished, the execution time statistics appear in the Profiler.
5. Click the "+" icons to expand rows in the Profiler to view the execution time statistics for the XSLT stylesheet (see *screenshot*). Note that in the case of these screenshots, **Hitcount Profiling** was selected.

Click on a row in the Profiler to highlight the corresponding instruction in the file that was analyzed.

The following screenshot shows the Tree View in the Profiler:

The screenshot displays the XSLT and XQuery Profiler interface. The top pane shows the XSLT code being analyzed, with the instruction `<xsl:for-each select='n1:Office'>` highlighted. The bottom pane shows the Tree View, which is a table listing the XSLT instructions and their performance metrics.

Name	Hit Count	Duration (ms)	%	Descendants (ms)	Descendants and Self...	%	XPath
xsl:stylesheet	1	3.69	1.17	311.83	315.52	99.99	0.00
xsl:output	0	0.00	0.00	0.00	0.00	0.00	0.00
xsl:template	1	0.02	0.01	311.81	311.83	98.82	0.00
html	1	0.11	0.03	311.70	311.81	98.82	0.00
head	1	0.06	0.02	0.05	0.11	0.03	0.00
body	1	0.04	0.01	311.55	311.60	98.75	0.00
xsl:for-each	1	0.22	0.07	311.33	311.55	98.74	0.18
xsl:for-each	1	0.18	0.06	1.30	1.49	0.47	0.17
br	1	0.04	0.01	0.00	0.04	0.01	0.00
xsl:for-each	1	0.19	0.06	0.78	0.97	0.31	0.18
br	1	0.04	0.01	0.00	0.04	0.01	0.00

The following screenshot shows List View in the Profiler for the same XSLT stylesheet:

The screenshot displays the XSLT and XQuery Profiler interface. The top section shows an XSLT stylesheet with the following code:

```
<xsl:for-each select="@href">
  <a>
    <xsl:attribute name="href"><xsl:text disable-output-escaping="yes">http:
    <img border="0">
      <xsl:attribute name="src"><xsl:value-of select="." /></xsl:attribute>
    </img>
  </a>
</xsl:for-each>
</div>
</xsl:for-each>
<br />
<xsl:for-each select="n1:Name">
  <span style="color:#0588BA; font-family:Arial; font-size:20pt; font-weight:bold; ">
    <xsl:apply-templates />
  </span>
</xsl:for-each>
<br />
<br />
<xsl:for-each select="n1:Office">
```

Below the code editor, there are tabs for "Text", "Grid", "Schema/WSDL", "Authentic", and "Browser". The "Text" tab is selected, showing the XSLT code. Below the tabs, there are icons for "OrgChart.xsl" and "OrgChart.html".

The "List" tab is selected, displaying a table of execution statistics:

Index	Name	Hit Count	Duration (ms)	%	Des...	Descendants an...	%	XPath
14	xsl:text	1	0.01	0.00	0.02	0.03	0.01	0.00
15	Text	1	0.02	0.01	0.00	0.02	0.01	0.00
16	img	1	0.04	0.01	0.78	0.82	0.26	0.00
17	border	1	0.02	0.01	0.00	0.02	0.01	0.00
18	xsl:attribute	1	0.10	0.03	0.66	0.76	0.24	0.00
19	xsl:value-of	1	0.66	0.21	0.00	0.66	0.21	0.58
20	br	1	0.04	0.01	0.00	0.04	0.01	0.00
21	xsl:for-each	1	0.19	0.06	0.78	0.97	0.31	0.18
22	span	1	0.06	0.02	0.72	0.78	0.25	0.00
23	style	1	0.02	0.01	0.00	0.02	0.01	0.00
24	xsl:apply-templates	1	0.60	0.19	0.10	0.70	0.22	0.15

At the bottom of the table, there are buttons for "Tree" and "List".

Using the Information in the Profiler

The Profiler displays the following information about each instruction in the XSLT stylesheet:

- **Index:** A number assigned to each instruction in the order in which the instruction was called.
- **Name:** The name of the XSLT instruction.
- **Hit Count:** The total number of times the instruction was called during the transformation.
- **Duration (ms) and %:** The number of milliseconds that the instruction took to execute without taking the execution time of its descendants into account, and the percentage of the total execution time.
- **Descendants (ms):** The number of milliseconds that the descendants of the instruction took to execute.

- **Descendants and Self** and **%**: The number of milliseconds that the instruction and its descendants took to execute, and the percentage of the total execution time.
- **XPath**: If the instruction contains an XPath statement, this column contains the time it took that statement to execute.

Please note: When using hitcount profiling, the times in the Profiler window are the sum total of execution time for all the hits to the instruction. When using callgraph profiling, because each call of the instruction is listed separately, the times shown in the Profiler window are the duration of a single execution of the instruction.

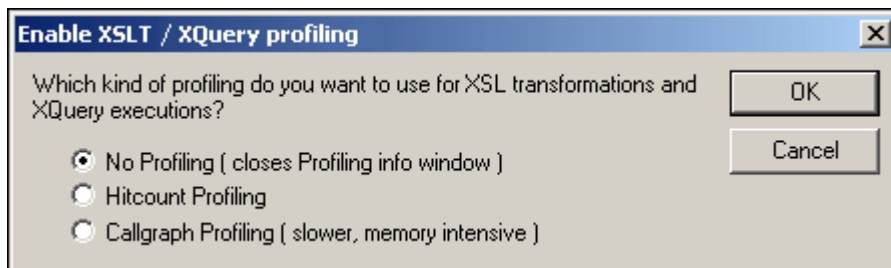
7.4.2 XQuery Profiling

Starting the Profiler

Please note: Execution time results displayed in the Profiler may be influenced by other applications that are running on your computer. When analyzing files using the Profiler, it is best to run only the XMLSpy application.

To analyze an XQuery document:

1. In XMLSpy, open the XQuery document that you want to analyze.
2. Activate the Profiler by selecting **XSL/XQuery | Enable XSLT 2 / XQuery profiling**. A dialog opens.



3. Select **Hitcount Profiling** or **Callgraph Profiling**. Click **OK** to confirm. An empty Profiler window appears.
4. Execute the XQuery (**XSL/XQuery | XQuery Execution**). When execution is finished, the execution time statistics appear in the Profiler.
5. Click the "+" icons to expand rows in the Profiler to view the execution time statistics for the instructions in the XQuery document (see *screenshot*). Note that in the case of these screenshots, **Hitcount Profiling** was selected.

Click on a row in the Profiler to highlight the corresponding instruction in the file that was analyzed.

The following screenshot shows the Tree View in the Profiler:

```

module namespace c="http://www.example.com/calc" ;
declare namespace ipo="http://www.example.com/IPO";

declare function c:total-price( $i as element()* )
as xs:decimal
{
  let $subtotals := for $s in $i return $s/quantity * $s/USPrice
  return sum( $subtotals )cast as xs:decimal
};

declare function c:quantity( $i as element()* )
as xs:decimal
{
  $i/quantity cast as xs:decimal
};

declare function c:price( $i as element()* )
as xs:decimal
{
  $i/USPrice cast as xs:decimal
};

```

Text

 strongQ11_callingfunction.xq
  OrgChart.xml
  XQuery Output.xml
  strongQ11.xq

Tree

Name	Info	Hit Count	Duration (ms)	%	Descendants and Self (ms)	%
[-] MainModule	strongQ11_callingfunction.xq					
[-] FLWORExpr	for \$p in doc("ipo.xml")//ele...	1	0.08	1.46	5.40	99.79
[-] LibraryModule	strongQ11.xq					
[-] Function	c:total-price(\$i as element()...	1	0.03	0.61	0.32	5.82
[-] Function	c:quantity(\$i as element()*)...	1	0.04	0.75	0.11	2.10
[-] Function	c:price(\$i as element()*) a...	1	0.03	0.63	0.10	1.79

Tree List

The following screenshot shows List View in the Profiler for the same XQuery document:

```

module namespace c="http://www.example.com/calc" ;
declare namespace ipo="http://www.example.com/IPO";
declare function c:quantity( $i as element()* )
  as xs:decimal
{
  $i/quantity cast as xs:decimal
}
declare function c:total-price( $i as element()* )
  as xs:decimal
{
  let $subtotals := for $s in $i return $s/quantity * $s/USPrice
  return sum( $subtotals )cast as xs:decimal
};

declare function c:price( $i as element()* )
  as xs:decimal
{
  $i/USPrice cast as xs:decimal
};
  
```

Text

strongQ11_callingfunction.xq OrgChart.xml XQuery Output.xml strongQ11.xq

List

Index	Name	Info	Hit Count	Duration (ms)	%	Descendants ...	%
59	SequenceType	element()*	1	0.01	0.21	0.02	0.31
58	TypeDeclaration	as element()*	1	0.00	0.07	0.02	0.38
57	Function	c:quantity(\$i as element()*)...	1	0.04	0.75	0.11	2.10
56	VarRef	\$subtotals	1	0.00	0.05	0.00	0.05
55	FunctionCall	sum(\$subtotals)	1	0.03	0.48	0.03	0.53
54	CastExpr	sum(\$subtotals)cast as xs...	1	0.07	1.32	0.10	1.84
53	NameTest	USPrice	1	0.00	0.08	0.00	0.08
52	VarRef	\$s	1	0.00	0.05	0.00	0.05
51	RelativePathExpr	\$s/USPrice	1	0.01	0.15	0.02	0.28

Tree List

Using the Information in the Profiler

The Profiler displays the following information about each instruction in the XQuery document:

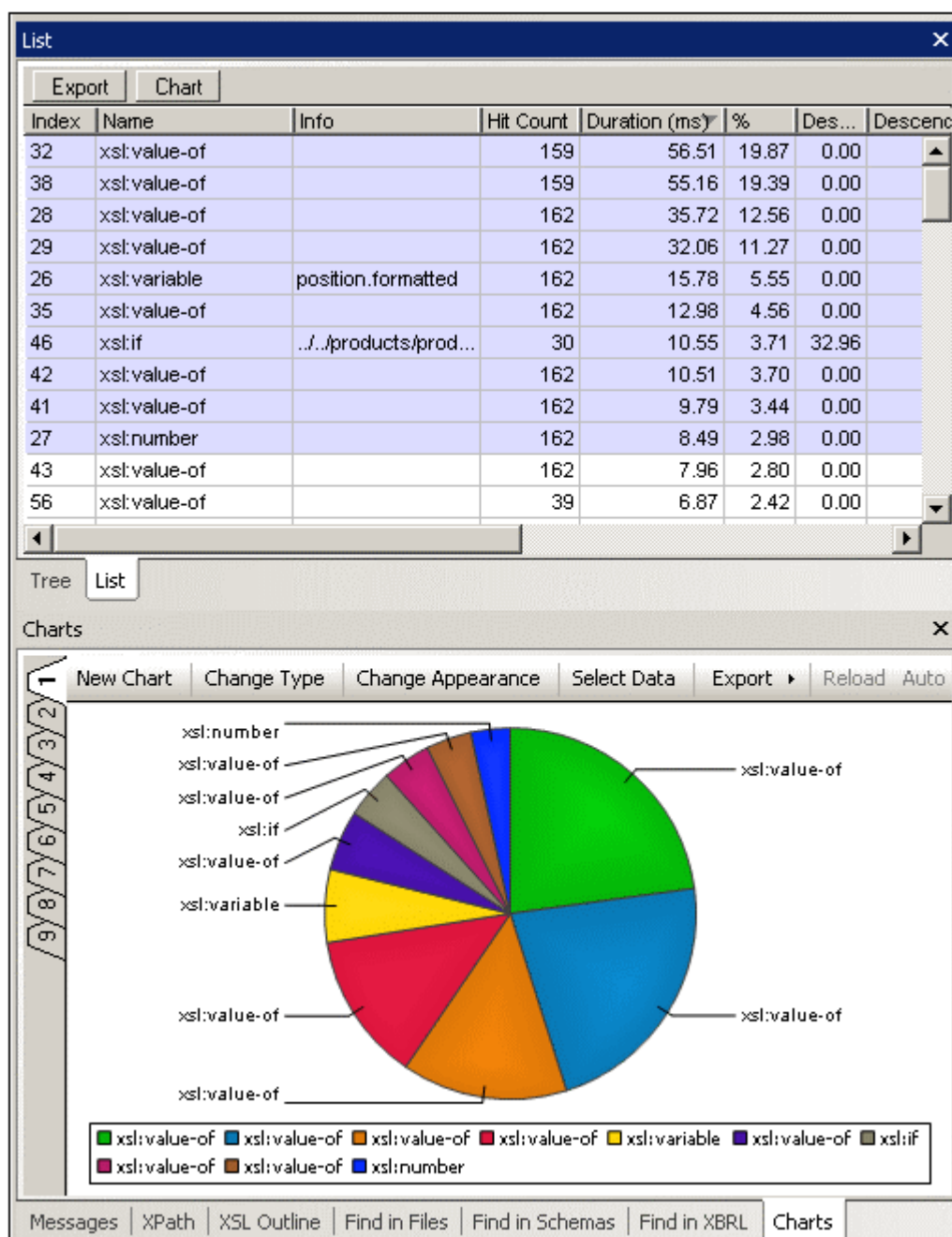
- **Index:** A number assigned to each instruction in the order in which the instruction was called.
- **Name:** The name of the XQuery instruction.
- **Info:** Information about the instruction. For example, if the instruction is a variable declaration, this column contains the name of the variable and its value; if it is a function, then this contains the name and parameters of the function.
- **Hit Count:** The total number of times the instruction was called during execution.

- **Duration (ms) and %:** The number of milliseconds that the instruction took to execute without taking the execution time of its descendants into account, and the percentage of the total execution time.
- **Descendants and Self (ms) and %:** The total time spent executing the instruction and its descendants, and the percentage of the total execution time.

Please note: When using hitcount profiling, the times in the Profiler window are the sum total of execution time for all the hits to the instruction. When using callgraph profiling, because each call of the instruction is listed separately, the times shown in the Profiler window are the duration of a single execution of the instruction.

7.4.3 Profiler Charts

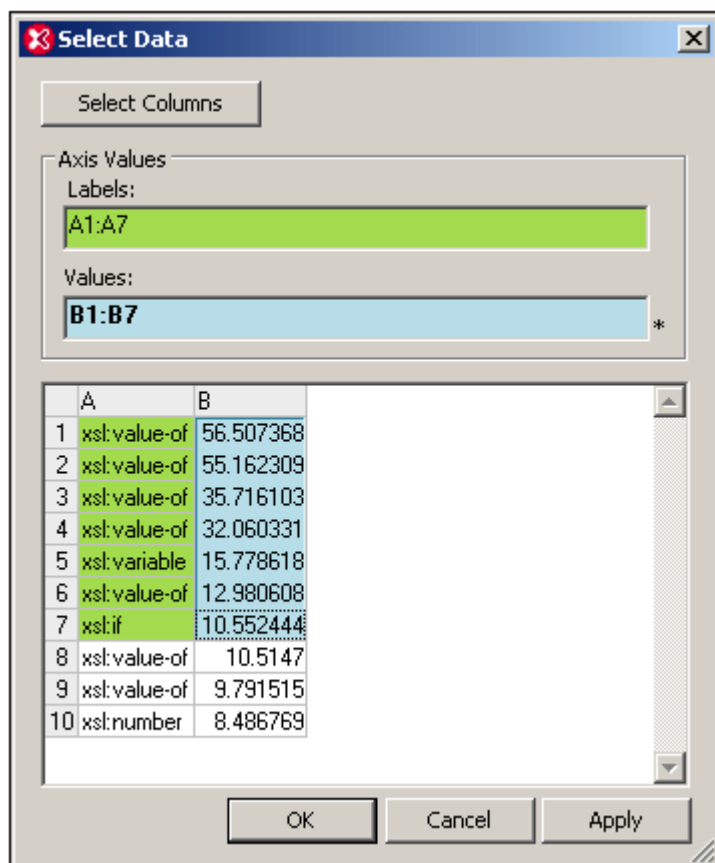
After running the XSLT/XQuery Profiler, a chart of the results or a subset of the results can be generated. In the Profiler window (*see screenshot below*), click the **Chart** button to generate the chart in the Charts output window (*see screenshot below*).



Note the following points:

- In the Profiler window (Tree View or List View), a subset of the results can be selected by marking them. Click with the **Ctrl** and/or **Shift** keys to mark multiple items. In List View the results can be sorted on the basis of a column's values by clicking on that column's header. This can be useful, for example, for ordering result items according to the most time-intensive items and then selecting a subset that takes up most of the transformation time. By selecting subsets unwanted result items can be filtered out. In the screenshot above, the highlighted result items have been selected.
- After a chart has been created its type (pie chart, bar chart, line chart, etc) can be changed by clicking the [Change Type](#) button of the Charts output window. The various types of charts are described in detail in the [Charts](#) section of the documentation.

- Clicking the **Select Data** button of the Charts output window pops up the Select Data dialog (screenshot below). In this dialog, you can select data for the X-Axis and Y-Axis from the data table that is produced by the Select Columns process. To select data for the X-Axis click in the Axis Values text box and then either enter the range of table values (for example, A1: A7) or drag the cursor from the start of the range to the end of the range. Do the same for the Y-Axis.



Clicking the **Select Columns** button enables you to change the data selection for the data table. See the [Source XPath](#), [X-Axis Selection](#), and [Y-Axis Selection](#) for information about how column selection works.

For more detailed information about charts see the [Charts](#) section of the documentation.

8 Authentic

Authentic View (*screenshot below*) is a graphical representation of your XML document. It enables XML documents to be displayed without markup and with appropriate formatting and data-entry features such as input fields, combo boxes, and radio buttons. Data that the user enters in Authentic View is entered into the XML file.

Nanonull, Inc.
 Location:

Street:	119 Oakstreet, Suite 4876	Phone:	+1 (321) 555 5155 0
City:	Vereno	Fax:	+1 (321) 555 5155 4
State & Zip:	DC 29213	E-mail:	office@nanonull.com

Vereno Office Summary: 4 departments, 15 employees.

The company was established **in Vereno in 1995** as a privately held software company. Since 1996, Nanonull has been actively involved in developing nanoelectronic software technologies. It released the first version of its acclaimed *NanoSoft Development Suite* in February 1999. Also in 1999, Nanonull increased its capital base with invesment from a consortium of private investment firms. The company has been expanding rapidly ever since.

To be able to view and edit an XML document in Authentic View, the XML document must be associated with a **StyleVision Power Stylesheet (SPS)**, which is created in Altova's StyleVision product. An SPS (.sps file) is, in essence, an XSLT stylesheet. It specifies an output presentation for an XML file that can include data-entry mechanisms. Authentic View users can, therefore, write data back to the XML file or DB. An SPS is based on a schema and is specific to it. If you wish to use an SPS to edit an XML file in Authentic View, you must use one that is based on the same schema as that on which the XML file is based.

Using Authentic View

- If an XML file is open, you can switch to Authentic View by clicking the **Authentic** button at the bottom of the Main Window. If an SPS is not already assigned to the XML file, you will be prompted to assign one to it. You must use an SPS that is based on the same schema as the XML file.
- A new XML file is created and displayed in Authentic View by selecting the **File | New** command and then clicking the "Select a StyleVision Stylesheet" button. This new file is a template file associated with the SPS you open. It can have a variable amount of starting data already present in it. This starting data is contained in an XML file (a Template XML File) that may optionally be associated with the SPS. After the Authentic View of an XML file is displayed, you can enter data in it and save the file.
- You can also open an SPS via the **Authentic | New Document** command. If a

Template XML File has been assigned to the SPS, then the data in the Template XML File is used as the starting data of the XML document template created in Authentic View.

In this section

This section contains an Authentic View tutorial, which shows you how to use Authentic View. It is followed by the section, Editing in Authentic View, which explains individual editing features in detail.

More information about Authentic View

For more information about Authentic View information, see (i) the section [Editing Views | Authentic View](#) in this documentation, which describes the Authentic View editing window, and (ii) the [Authentic menu](#) section of the User Reference part of this documentation.

8.1 Authentic View Tutorial

In Authentic View, you can edit XML documents in a graphical WYSIWYG interface (*screenshot below*), just like in word-processor applications such as Microsoft Word. In fact, all you need to do is enter data. You do not have to concern yourself with the formatting of the document, since the formatting is already defined in the stylesheet that controls the Authentic View of the XML document. The stylesheet (StyleVision Power Stylesheet, shortened to SPS in this tutorial) is created by a stylesheet designer using Altova's StyleVision product.

Nanonull, Inc.	
Location: US	
Street:	119 Oakstreet, Suite 4876
City:	Vereno
State & Zip:	DC 29213
Phone:	+1 (321) 555 5155 0
Fax:	+1 (321) 555 5155 4
E-mail:	office@nanonull.com

Vereno Office Summary: 4 departments, 15 employees.

The company was established **in Vereno in 1995** as a privately held software company. Since 1996, Nanonull has been actively involved in developing nanoelectronic software technologies. It released the first version of its acclaimed *NanoSoft Development Suite* in February 1999. Also in 1999, Nanonull increased its capital base with investment from a consortium of private investment firms. The company has been expanding rapidly ever since.

Editing an XML document in Authentic View involves two user actions: (i) editing the structure of the document (for example, adding or deleting document parts, such as paragraphs and headlines); and (ii) entering data (the content of document parts).

This tutorial takes you through the following steps:

- Opening an XML document in Authentic View. The key requirement for Authentic View editing is that the XML document be associated with an SPS file.
- A look at the Authentic View interface and a broad description of the central editing mechanisms.
- Editing document structure by inserting and deleting nodes.
- Entering data in the XML document.
- Entering (i) attribute values via the Attributes entry helper, and (ii) entity values.
- Printing the document.

Remember that this tutorial is intended to get you started, and has intentionally been kept simple. You will find additional reference material and feature descriptions in the [Authentic View interface](#) section.

Tutorial requirements

All the files you need for the tutorial are in the `C:\Documents and Settings\<username>\My Documents\Altova\<ProductName>\Examples` folder of your Altova application folder. These files are:

- `NanonullOrg.xml` (the XML document you will open)
- `NanonullOrg.sps` (the StyleVision Power Stylesheet to which the XML document is linked)
- `NanonullOrg.xsd` (the XML Schema on which the XML document and StyleVision Power Stylesheet are based, and to which they are linked)
- `nanonull.gif` and `Altova_right_300.gif` (two image files used in the tutorial)

Please note: At some points in the tutorial, we ask you to look at the XML text of the XML document (as opposed to the Authentic View of the document). If the Altova product edition you are using does not include a Text View (as with Authentic Desktop and Authentic Browser), then use a plain **text editor** like Wordpad or Notepad to view the text of the XML document.

Caution: We recommend that you use a copy of `NanonullOrg.xml` for the tutorial, so that you can always retrieve the original should the need arise.

8.1.1 Opening an XML Document in Authentic View

In Authentic View, you can edit an existing XML document or create and edit a new XML document. In this tutorial, you will open an existing XML document in Authentic View (described in this section) and learn how you can edit it (subsequent sections). Additionally, in this section is a description of how a new XML document can be created for editing in Authentic View.

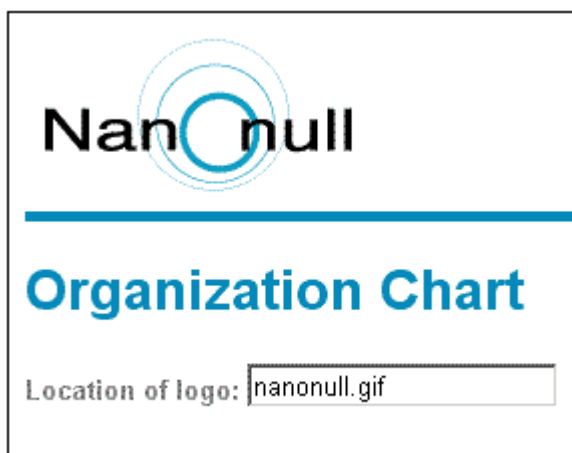
Opening an existing XML document

The file you will open is `NanonullOrg.xml`. It is in the `Examples` folder of your Altova application. You can open `NanonullOrg.xml` in one of two ways:

- Click **File | Open** in your Altova product, then browse for `NanonullOrg.xml` in the dialog that appears, and click **Open**.
- Use Windows Explorer to locate the file, right-click, and select your Altova product as the application with which to open the file.

The file `NanonullOrg.xml` opens directly in Authentic View (*screenshot below*). This is because

1. The file already has a StyleVision Power Stylesheet (SPS) assigned to it, and
2. In the Options dialog (**Tools | Options**), in the View tab, the option to open XML files in Authentic View if an SPS file is assigned has been checked. (Otherwise the file would open in Text View.)



Remember: It is the SPS that defines and controls how an XML document is displayed in Authentic View. Without an SPS, there can be no Authentic View of the document.

Creating a new XML document based on an SPS

You can also create a new XML document that is based on an SPS. You can do this in two ways: via the **File | New** menu command and via the **Authentic | New Document** menu command. In both cases an SPS is selected.

Via File | New

1. Select **File | New**, and, in the Create a New Document dialog, select XML as the new file type to create.
2. Click **Select a STYLEVISION Stylesheet**, and browse for the desired SPS.

Via Authentic | New Document

1. Select **Authentic | New Document**.
2. In the Create a New Document dialog, browse for the desired SPS.

If a Template XML File has been assigned to the SPS, then the data in the Template XML File is used as the starting data of the XML document template created in Authentic View.

8.1.2 The Authentic View Interface

The Authentic View editing interface consists of a main window in which you enter and edit the document data, and three entry helpers. Editing a document is simple. If you wish to see the markup of the document, switch on the markup tags. Then start typing in the content of your document. To modify the document structure, you can use either the context menu or the Elements entry helper.

Displaying XML node tags (document markup)

An XML document is essentially a hierarchy of nodes. For example:

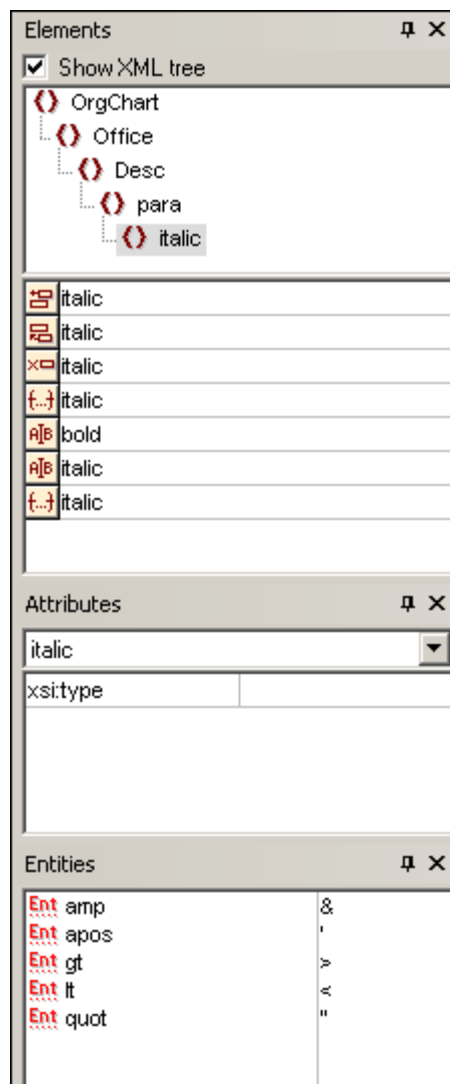
```
<DocumentRoot>
  <Person id="ABC001">
    <Name>Alpha Beta</Name>
    <Address>Some Address</Address>
    <Tel>1234567</Tel>
  </Person>
</DocumentRoot>
```

By default, the node tags are not displayed in Authentic View. You can switch on the node tags by selecting the menu item **Authentic | Show Large Markup** (or the  toolbar icon). Large markup tags contain the names of the respective nodes. Alternatively, you can select small markup (no node names in tags) and mixed markup (a mixture of large, small, and no markup tags, which is defined by the designer of the stylesheet; the default mixed markup for the document is no markup).

You can view the text of the XML document in the Text View of your Altova product or in a text editor.

Entry helpers

There are three entry helpers in the interface (*screenshot below*), located by default along the right edge of the application window. These are the Elements, Attributes, and Entity entry helpers.



Elements entry helper: The Elements entry helper displays elements that can be inserted and removed with reference to the current location of the cursor or selection in the Main Window.

Note that the entry helper is context-sensitive; its content changes according to the location of the cursor or selection. The content of the entry helper can be changed in one other way: when another node is selected in the XML tree of the Elements entry helper, the elements relevant to that node are displayed in the entry helper. The Elements entry helper can be expanded to show the XML tree by checking the Show XML Tree check box at the top of the entry helper (*see screenshot above*). The XML tree shows the hierarchy of nodes from the top-level element node all the way down to the node selected in the Main Window.

Attributes entry helper: The Attributes entry helper displays the attributes of the element selected in the Main Window, and the values of these attributes. Attribute values can be entered or edited in the Attributes entry helper. Element nodes from the top-level element down to the selected element are available for selection in the combo box of the Attributes entry helper. Selecting an element from the dropdown list of the combo box causes that element's attributes to be displayed in the entry helper, where they can then be edited.

Entities entry helper: The Entities entry helper is not context-sensitive, and displays all the entities declared for the document. Double-clicking an entity inserts it at the cursor location. How to add entities for a document is described in the section [Authentic View Interface](#).

Context menu

Right-clicking at a location in the Authentic View document pops up a context menu relevant to that (node) location. The context menu provides commands that enable you to:

- Insert nodes at that location or before or after the selected node. Submenus display lists of nodes that are allowed at the respective insert locations.
- Remove the selected node (if this allowed by the schema) or any removable ancestor element. The nodes that may be removed (according to the schema) are listed in a submenu.
- Insert entities and CDATA sections. The entities declared for the document are listed in a submenu. CDATA sections can only be inserted within text.
- Cut, copy, paste (including pasting as XML or text), and delete document content.

Note: For more details about the interface, see [Authentic View Interface](#)

8.1.3 Node Operations

There are two major types of nodes you will encounter in an Authentic View XML document: **element nodes** and **attribute nodes**. These nodes are marked up with tags, which you can [switch on](#). There are also other nodes in the document, such as text nodes (which are not marked up) and CDATA section nodes (which are marked up, in order to delimit them from surrounding text).

The node operations described in this section refer only to element nodes and attribute nodes. When trying out the operations described in this section, it is best to have [large markup switched on](#).

Note: It is important to remember that **only same- or higher-level elements** can be inserted before or after the selected element. Same-level elements are **siblings**. Siblings of a paragraph element would be other paragraph elements, but could also be lists, a table, an image, etc. Siblings could occur before or after an element. Higher-level elements are **ancestor** elements and siblings of ancestors. For a paragraph element, ancestor elements could be a section, chapter, article, etc. A paragraph in a valid XML file would already have ancestors. Therefore, adding a higher-level element in Authentic View, creates the new element as a sibling of the relevant ancestor. For example, if a section

element is inserted after a paragraph, it is created as a sibling of the section that contains the current paragraph element.

Carrying out node operations

Node operations can be carried out by selecting a command in the [context menu](#) or by clicking the node operation entry in the [Elements entry helper](#). In some cases, an element or attribute can be added by clicking the [Add Node link](#) in the Authentic View of the document. In the special cases of elements defined as paragraphs or list items, pressing the [Enter key](#) when within such an element creates a new sibling element of that kind. This section also describes how nodes can be created and deleted by using the [Apply Element](#), [Remove Node](#), and [Clear Element](#) mechanisms.

Inserting elements

Elements can be inserted at the following locations:

- The cursor location within an element node. The elements available for insertion at that location are listed in a submenu of the context menu's **Insert** command. In the Elements entry helper, elements that can be inserted at a location are indicated with the  icon. In the `NanonullOrg.xml` document, place the cursor inside the `para` element, and create `bold` and `italic` elements using both the context menu and Elements entry helper.
- Before or after the selected element or any of its ancestors, if allowed by the schema. Select the required element from the submenu/s that roll out. In the Elements entry helper, elements that can be inserted before or after the selected element are indicated with the  and  icons, respectively. Note that in the Elements entry helper, you can insert elements before/after the selected element only; you cannot insert before/after an ancestor element. Try out this command, by first placing the cursor inside the `para` element and then inside the table listing the employees.

Add Node link

If an element or attribute is included in the document design, and is not present in the XML document, an `Add Node` link is displayed at the location in the document where that node is specified. To see this link, in the line with the text, *Location of logo*, select the `@href` node within the `CompanyLogo` element and delete it (by pressing the **Delete** key). The `add @href` link appears within the `CompanyLogo` element that was edited (*screenshot below*). Clicking the link adds the `@href` node to the XML document. The text box within the `@href` tags appears because the design specifies that the `@href` node be added like this. You still have to enter the value (or content) of the `@href` node. Enter the text `nanonull.gif`.



If the content model of an element is ambiguous, for example, if it specifies that a sequence of child elements may appear in any order, then the `add...` link appears. Note that no node name

is specified. Clicking the link will pop up a list of elements that may validly be inserted.

Note: The Add Node link appears directly in the document template; there is no corresponding entry in the context menu or Elements entry helper.

Creating new elements with the Enter key

In cases where an element has been formatted as a paragraph or list item (by the stylesheet designer), pressing the Enter key when inside such a node causes a new node of that kind to be inserted after the current node. You can try this mechanism in the `NanonullOrg.xml` document by going to the end of a `para` node (just before its end tag) and pressing **Enter**.

Applying elements

In elements of mixed content (those which contain both text and child elements), some text content can be selected and an allowed child element be applied to it. The selected text becomes the content of the applied element. To apply elements, in the context menu, select **Apply** and then select from among the applicable elements. (If no elements can be applied to the selected text, then the **Apply** command does not appear in the context menu.) In the Elements entry helper, elements that can be applied for a selection are indicated with the  icon. In the `NanonullOrg.xml` document, select text inside the mixed content `para` element and experiment with applying the `bold` and `italic` elements.

The stylesheet designer might also have created a toolbar icon to apply an element. In the `NanonullOrg.xml` document, the `bold` and `italic` elements can be applied by clicking the `bold` and `italic` icons in the application's Authentic toolbar.

Removing nodes

A node can be removed if its removal does not render the document invalid. Removing a node causes a node and all its contents to be deleted. A node can be removed using the **Remove** command in the context menu. When the **Remove** command is highlighted, a submenu pops up which contains all nodes that may be removed, starting from the selected node and going up to the document's top-level node. To select a node for removal, the cursor can be placed within the node, or the node (or part of it) can be highlighted. In the Elements entry helper, nodes that can be removed are indicated with the  icon. A removable node can also be removed by selecting it and pressing the **Delete** key. In the `NanonullOrg.xml` document, experiment with removing a few nodes using the mechanisms described. You can undo your changes with **Ctrl+Z**.

Clearing elements

Element nodes that are children of elements with mixed content (both text and element children) can be cleared. The entire element can be cleared when the node is selected or when the cursor is placed inside the node as an insertion point. A text fragment within the element can be cleared of the element markup by highlighting the text fragment. With the selection made, select **Clear** in the context menu and then the element to clear. In the Elements entry helper, elements that can be cleared for a particular selection are indicated with the  icon (insertion point selection) and  icon (range selection). In the `NanonullOrg.xml` document, try the clearing mechanism with the `bold` and `italic` child elements of `para` (which has mixed content).

Tables and table structure

There are two types of Authentic View table:

- *SPS tables (static and dynamic)*. The broad structure of SPS table is determined by the stylesheet designer. Within this broad structure, the only structural changes you are allowed are content-driven. For example, you could add new rows to a dynamic SPS table.
- *XML tables*, in which you decide to present the contents of a particular node (say, one for person-specific details) as a table. If the stylesheet designer has enabled the creation of this node as an XML table, then you can determine the structure of the table and edit its contents. XML tables are discussed in detail in the [Using tables in Authentic View](#) section.

8.1.4 Entering Data in Authentic View

Data is entered into the XML document directly in the main window of Authentic View. Additionally for attributes, data (the value of the attribute) can be [entered in the Attributes entry helper](#). Data is entered (i) directly as text, or (ii) by selecting an option in a data-entry device, which is then mapped to a predefined text entry.

Adding text content

You can enter element content and attribute values directly as text in the main window of Authentic View. To insert content, place the cursor at the location where you want to insert the text, and type. You can also copy text from the clipboard into the document. Content can also be edited using standard editing mechanisms, such as the **Caps** and **Delete** keys. For example, you can highlight the text to be edited and type in the replacement text with the **Caps** key on.

For example, to change the name of the company, in the `Name` field of `Office`, place the cursor after Nanonull, and type in `USA` to change the name from Nanonull, Inc. to Nanonull USA, Inc.



If text is editable, you will be able to place your cursor in it and highlight it, otherwise you will not be able to. Try changing any of the **field names** (not the field values), such as "Street", "City", or "State/Zip," in the address block. You are not able to place the cursor in this text because such text is not XML content; it is derived from the StyleVision Power Stylesheet.

Inserting special characters and entities

When entering data, the following type of content is handled in a special way:

- *Special characters that are used for XML markup* (ampersand, apostrophe, greater than, less than, and quotes). These characters are available as [built-in entities](#) and can be entered in the document by double-clicking the respective entity in the Entities entry helper. If these characters occur frequently (for example, in program code listings), then they can be entered within CDATA sections. To insert a CDATA section, right-click at the location where you wish to enter the CDATA section, and select **Insert CDATA Section** from the context menu. The XML processor ignores all markup characters within CDATA sections. This also means that if you want a special character inside a CDATA section, you should enter that character and not its entity reference.
- *Special characters that cannot be entered via the keyboard* should be entered by copying them from the character map of your system to the required location in the document.

- A frequently used text string can be [defined as an entity](#), which appears in the Entities entry helper. The [entity is inserted](#) at the required locations by placing the cursor at each required location and double-clicking the entity in the entry helper. This is useful for maintenance because the value of the text string is held in one location; if the value needs to be changed, then all that needs to be done is to change the entity definition.

Note: When markup is hidden in Authentic View, an empty element can easily be overlooked. To make sure that you are not overlooking an empty element, [switch large or small markup on](#).

Try using each type of text content described above.

Adding content via a data-entry device

In the content editing you have learned above, content is added by directly typing in text as content. There is one other way that **element content** (or attribute values) can be entered in Authentic View: via data-entry devices.

Given below is a list of data-entry devices in Authentic View, together with an explanation of how data is entered in the XML file for each device.

Data-Entry Device	Data in XML File
Input Field (Text Box)	Text entered by user
Multiline Input Field	Text entered by user
Combo box	User selection mapped to value
Check box	User selection mapped to value
Radio button	User selection mapped to value
Button	User selection mapped to value

In the static table containing the address fields (*shown below*), there are two data-entry devices: an input field for the `zip` field and a combo-box for the State field. The values that you enter in the text fields are entered directly as the XML content of the respective elements. For other data-entry devices, your selection is mapped to a value.

For the Authentic View shown above, here is the corresponding XML text:

```
<Address>
  <ipo:street>119 oakstreet, Suite 4876</ipo:street>
  <ipo:city>Vereno</ipo:city>
  <ipo:state>DC</ipo:state>
  <ipo:zip>29213</ipo:zip>
</Address>
```

Notice that the combo-box selection `DC` is mapped to a value of `DC`. The value of the `zip` field is entered directly as content of the `ipo: zip` element.

8.1.5 Entering Attribute Values

An attribute is a property of an element, and an element can have any number of attributes. Attributes have values. You may sometimes be required to enter XML data as an attribute value. In Authentic View, you enter attribute values in two ways:

- As content in the main window if the attribute has been created to accept its value in this way
- In the Attributes entry helper

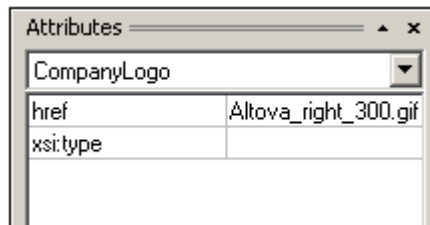
Attribute values in the main window

Attribute values can be entered as normal text or as text in an input field, or as a user selection that will be mapped to an XML value. They are entered in the same way that element content is entered: see [Entering Data in Authentic View](#). In such cases, the distinction between element content and attribute value is made by the StyleVision Power Stylesheet and the data is handled appropriately.

Attribute values in the Attributes Entry Helper

If you wish to enter or change an attribute value, you can also do this in the Attributes Entry Helper. First, the attribute node is selected in Authentic View, then the value of the attribute is entered or edited in the Attributes entry helper. In the `NanonullOrg.xml` document, the location of the logo is stored as the value of the `href` attribute of the `CompanyLogo` element. To change the logo to be used:

1. Select the `CompanyLogo` element by clicking a `CompanyLogo` tag. The attributes of the `CompanyLogo` element are displayed in the Attributes Entry Helper.
2. In the Attributes Entry Helper, change the value of the `href` attribute from `nanonull.gif` to `Altova_right_300.gif` (an image in the `Examples` folder).

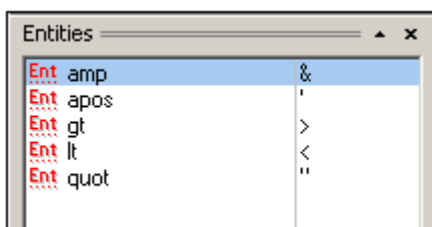


This causes the Nanonull logo to be replaced by the Altova logo.

Note: Entities cannot be entered in the Attributes entry helper.

8.1.6 Adding Entities

An entity in Authentic View is typically XML data (but not necessarily), such as a single character; a text string; and even a fragment of an XML document. An entity can also be a binary file, such as an image file. All the entities available for a particular document are displayed in the Entities Entry Helper (*screenshot below*). To insert an entity, place the cursor at the location in the document where you want to insert it, and then double-click the entity in the Entities entry helper. Note that you cannot enter entities in the Attributes entry helper.



The ampersand character (&) has special significance in XML (as have the apostrophe, less than and greater than symbols, and the double quote). To insert these characters, entities are used so that they are not confused with XML-significant characters. These characters are available as entities in Authentic View.

In `NanonullOrg.xml`, change the title of Joe Martin (in Marketing) to Marketing Manager Europe & Asia. Do this as follows:

1. Place the cursor where the ampersand is to be inserted.
2. Double-click the entity listed as "amp". This inserts an ampersand (*screenshot below*).

Marketing (2)		
First	Last	Title
Joe	Martin	Marketing Manager Europe &

Note: The Entities Entry Helper is not context-sensitive. All available entities are displayed no matter where the cursor is positioned. This does not mean that an entity can be inserted at all locations in the document. If you are not sure, then validate the document after inserting the entity: **XML | Validate XML (F8)**.

Defining your own entities

As a document editor, you can define your own document entities. How to do this is described in the section [Defining Entities in Authentic View](#).

8.1.7 Printing the Document

A printout from Authentic View of an XML document preserves the formatting seen in Authentic View.

To print `NanonullOrg.xml`, do the following:

1. Switch to Hide Markup mode if you are not already in it. You must do this if you do not want markup to be printed.
2. Select **File | Print Preview** to see a preview of all pages. Shown below is part of a print preview page, reduced by 50%.

Page 1 of 1

ALTOVA

www.altova.com

Organization Chart

Location of logo: `Altova_right_300.gif`

Nanonull, Inc.

Location: `US`

Street: 119 Calzad, Suite 4076	Phone: +1 (22) 555 555 0	
City: Mexico	Fax: +1 (22) 555 555 4	
State & Zip: <code>EC</code> <code>2220</code>	E-mail: office@nanonull.com	

[View Client History](#) | [Download Info](#) | [Terms of Use](#)

The company was established in Mexico in 1995 as a privately held software company. Since 1995, Nanonull has been actively involved in developing customized software solutions. It released the first version of its software development Suite in February 1997. Version 950, Nanonull has also signed a distribution agreement with a major computer manufacturer. The company is now expanding its product line for development.

Notice that the formatting of the page is the same as that in Authentic View.

3. To print the file, click **File | Print**.

Note that you can also print a version of the document that displays markup. To do this, switch Authentic View to Show small markup mode or Show large markup mode, and then print.

8.2 Editing in Authentic View

This section describes important features of Authentic View in detail. Features have been included in this section either because they are frequently used or because the mechanisms or concepts involved require explanation.

The section explains the following:

- There are three distinct types of tables used in Authentic View. The section [Using tables in Authentic View](#) explains the three types of tables (static SPS, dynamic SPS, and XML), and when and how to use them. It starts with the broad, conceptual picture and moves to the details of usage.
- The Date Picker is a graphical calendar that enters dates in the correct XML format when you click a date. See [Using the Date Picker](#).
- An entity is shorthand for a special character or text string. You can define your own entities, which allows you to insert these special characters or text strings by inserting the corresponding entities. See [Defining Entities](#) for details.
- In the Enterprise and Professional editions of Altova products, Authentic View users can sign XML documents with [digital XML signatures](#) and verify these signatures.
- What [image formats](#) can be displayed in Authentic View.

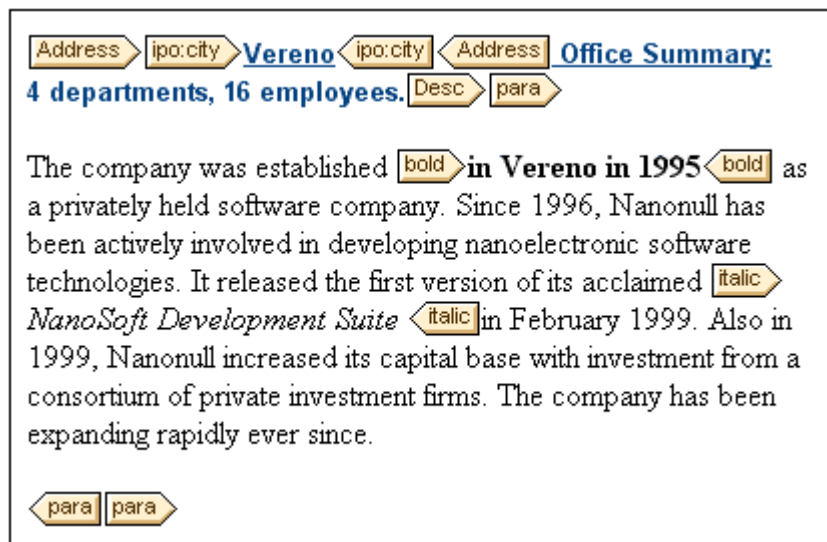
8.2.1 Basic Editing

When you edit in Authentic View, you are editing an XML document. Authentic View, however, can hide the structural XML markup of the document, thus displaying only the content of the document (*first screenshot below*). You are therefore not exposed to the technicalities of XML, and can edit the document as you would a normal text document. If you wish, you could switch on the markup at any time while editing (*second screenshot below*).

Vereno Office Summary: 4 departments, 16 employees.

The company was established **in Vereno in 1995** as a privately held software company. Since 1996, Nanonull has been actively involved in developing nanoelectronic software technologies. It released the first version of its acclaimed *NanoSoft Development Suite* in February 1999. Also in 1999, Nanonull increased its capital base with investment from a consortium of private investment firms. The company has been expanding rapidly ever since.

An editable Authentic View document with no XML markup.

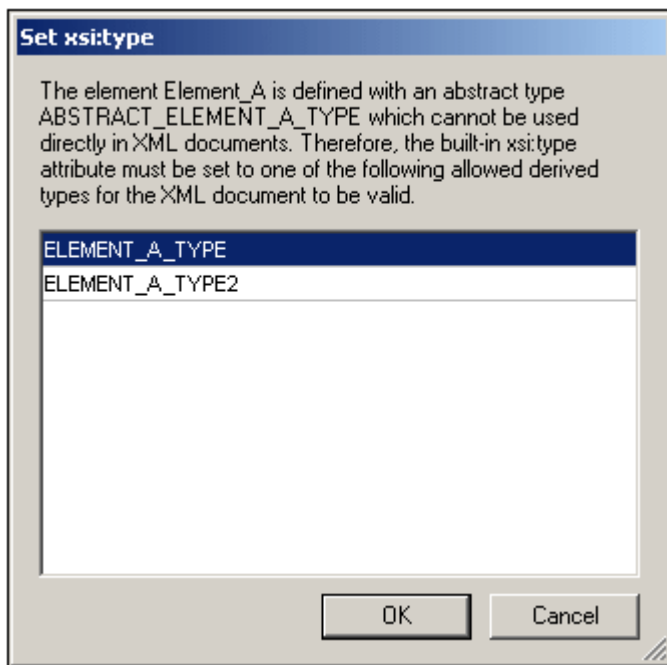


An editable Authentic View document with XML markup tags.

Inserting nodes

Very often you will need to add a new node to the Authentic XML document. For example, a new `Person` element might need to be added to an address book type of document. In such cases the XML Schema would allow the addition of the new element. All you need to do is right-click the node in the Authentic View document before which or after which you wish to add the new node. In the context menu that appears, select **Insert Before** or **Insert After** as required. The nodes available for insertion at that point in the document are listed in a submenu. Click the required node to insert it. The node will be inserted. All mandatory descendant nodes are also inserted. If a descendant node is optional, a clickable link, [Add NodeName](#), appears to enable you to add the optional node if you wish to.

If the node being added is an element with an abstract type, then a dialog (*something like in the screenshot below*) appears containing a list of derived types that are available in the XML Schema.



Selecting one of the available derived types and clicking **OK** does the following:

- Sets the selected derived type as the value of the `xsi:type` attribute of the element
- Inserts the element together with the descendant nodes defined in the content model of the selected derived type.

The selected derived type can be changed subsequently by changing the value of the element's `xsi:type` attribute in the Attributes Entry Helper. When the element's type is changed in this way, all nodes of the previous type's content model are removed and nodes of the new type's content model are inserted.

Text editing

An Authentic View document will essentially consist of text and images. To edit the text in the document, place the cursor at the location where you wish to insert text, and type. You can copy, move, and delete text using familiar keystrokes (such as the **Delete** key) and drag-and-drop mechanisms. One exception is the **Enter** key. Since the Authentic View document is pre-formatted, you do not—and cannot—add extra lines or space between items. The **Enter** key in Authentic View therefore serves to append another instance of the element currently being edited, and should be used exclusively for this purpose.

Copy as XML or as text

Text can be copied and pasted as XML or as text.

- If text is pasted as XML, then the XML markup is pasted together with the text content of nodes. The XML markup is pasted even if only part of a node's contents has been copied. For the markup to be pasted it must be allowed, according to the schema, at the location where it is pasted.
- If text is pasted as text, XML markup is not pasted.

To paste as XML or text, first copy the text (**Ctrl+C**), right-click at the location where the text is to be pasted, and select the context menu command **Paste As | XML** or **Paste As | Text**. If the shortcut **Ctrl+V** is used, the text will be pasted in the default Paste Mode of the SPS. The default Paste Mode will have been specified by the designer of the SPS. For more details, see

the section [Context Menus](#).

Alternatively, highlighted text can be dragged to the location where it is to be pasted. When the text is dropped, a pop-up appears asking whether the text is to be pasted as text or XML. Select the desired option.

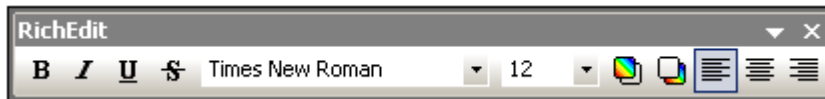
Text formatting

A fundamental principle of XML document systems is that content be kept separate from presentation. The XML document contains the content, while the stylesheet contains the presentation (formatting). In Authentic View, the XML document is presented via the stylesheet. This means that all the formatting you see in Authentic View is produced by the stylesheet. If you see bold text, that bold formatting has been provided by the stylesheet. If you see a list or a table, that list format or table format has been provided by the stylesheet. The XML document, which you edit in Authentic View contains only the content; it contains no formatting whatsoever. The formatting is contained in the stylesheet. What this means for you, the Authentic View user, is that you do not have to—nor can you—format any of the text you edit. You are editing content. The formatting that is automatically applied to the content you edit is linked to the semantic and/or structural value of the data you are editing. For example, an email address (which could be considered a semantic unit) will be formatted automatically in a certain way because of it is an email. In the same way, a headline must occur at a particular location in the document (both a structural and semantic unit) and will be formatted automatically in the way the stylesheet designer has specified that headlines be formatted. You cannot change the formatting of either email address or headline. All that you do is edit the content of the email address or headline.

In some cases, content might need to be specially presented; for example, a text string that must be presented in boldface. In all such cases, the presentation must be tied in with a structural element of the document. For example, a text string that must be presented in boldface, will be structurally separated from surrounding content by markup that the stylesheet designer will format in boldface. If you, as the Authentic View user, need to use such a text string, you would need to enclose the text string within the appropriate element markup. For information about how to do this, see the Insert Element command in the [Elements Entry Helper](#) section of the documentation.

Using RichEdit in Authentic View

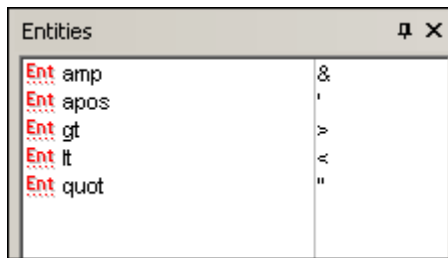
In Authentic View, when the cursor is placed inside an element that has been created as a RichEdit component, the buttons and controls in the RichEdit toolbar (*screenshot below*) become enabled. Otherwise they are grayed out.



Select the text you wish to style and specify the styling you wish to apply via the buttons and controls of the RichEdit toolbar. RichEdit enables the Authentic View user to specify the font, font-weight, font-style, font-decoration, font-size, color, background color and alignment of text. The text that has been styled will be enclosed in the tags of the styling element.

Inserting entities

In XML documents, some characters are reserved for markup and cannot be used in normal text. These are the ampersand (&), apostrophe ('), less than (<), greater than (>), and quote (") characters. If you wish to use these characters in your data, you must insert them as entity references, via the [Entities Entry Helper](#) (*screenshot below*).



XML also offers the opportunity to create custom entities. These could be: (i) special characters that are not available on your keyboard, (ii) text strings that you wish to re-use in your document content, (iii) XML data fragments, or (iv) other resources, such as images. You can [define your own entities](#) within the Authentic View application. Once defined, these entities appear in the [Entities Entry Helper](#) and can then be inserted as in the document.

Inserting CDATA sections

CDATA sections are sections of text in an XML document that the XML parser does not process as XML data. They can be used to escape large sections of text if replacing special characters by entity references is undesirable; this could be the case, for example, with program code or an XML fragment that is to be reproduced with its markup tags. CDATA sections can occur within element content and are delimited by `<![CDATA[` and `]]>` at the start and end, respectively. Consequently the text string `]]>` should not occur within a CDATA section as it would prematurely signify the end of the section. In this case, the greater than character should be escaped by its entity reference (`>`). To insert a CDATA section within an element, place the cursor at the desired location, right-click, and select **Insert CDATA Section** from the context menu. To see the CDATA section tags in Authentic View, [switch on the markup display](#). Alternatively, you could highlight the text that is to be enclosed in a CDATA section, and then select the **Insert CDATA section** command.

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

Editing and following links

A hyperlink consists of two parts: the link text and the target of the link. You can edit the link text by clicking in the text and editing. But you cannot edit the target of the link. (The target of the link is set by the designer of the stylesheet (either by typing in a static target address or by deriving the target address from data contained in the XML document).) From Authentic View, you can go to the target of the link by pressing **Ctrl** and clicking the link text. (Remember: merely clicking the link will set you up for editing the link text.)

8.2.2 Tables in Authentic View

The three table types fall into two categories: SPS tables (static and dynamic) and CALS/HTML Tables.

SPS tables are of two types: static and dynamic. SPS tables are designed by the designer of the StyleVision Power Stylesheet to which your XML document is linked. You yourself cannot insert an SPS table into the XML document, but you can enter data into SPS table fields and add and delete the rows of dynamic SPS tables. The section on [SPS tables](#) below explains the features of these tables.

CALS/HTML tables are inserted by you, the user of Authentic View. Their purpose is to enable you to insert tables at any allowed location in the document hierarchy should you wish to do so.

The editing features of [CALS/HTML Tables](#) and the [CALS/HTML Table editing icons](#) are described below.

SPS Tables

Two types of SPS tables are used in Authentic View: static tables and dynamic tables.

Static tables are fixed in their structure and in the content-type of cells. You, as the user of Authentic View, can enter data into the table cells but you cannot change the structure of these tables (i.e. add rows or columns, etc) or change the content-type of a cell. You enter data either by typing in text, or by selecting from options presented in the form of check-box or radio button alternatives or as a list in a combo-box. After you enter data, you can edit it.

Nanonull, Inc.		
Street:	119 Oakstreet, Suite 4876	Phone: +1 (321) 555 5155
City:	Vereno	Fax: +1 (321) 555 5155 - 9
State & Zip:	DC 29213	E-mail: office@nanonull.com

Please note: The icons or commands for editing dynamic tables **must not** be used to edit static tables.

Dynamic tables have rows that represent a repeating data structure, i.e. each row has an identical data structure (not the case with static tables). Therefore, you can perform row operations: append row, insert row, move row up, move row down, and delete row. These commands are available under the **Authentic** menu and as icons in the toolbar (shown below).



To use these commands, place the cursor anywhere in the appropriate row, and then select the required command.

Administration

First	Last	Title	Ext	EMail	Shares	Leave		
						Total	Used	Left
Vernon	Callaby	Office Manager	581	v.callaby@nanonull.com	1500	25	4	21
Frank	Further	Accounts Receivable	471	f.further@nanonull.com	0	22	2	20
Loby	Matise	Accounting Manager	963	l.matise@nanonull.com	add Shares	25	7	18
Employees: 3 (20% of Office, 9% of Company)					Shares: 1500 (13% of Office, 6% of Company)			
Non-Shareholders: Frank Further, Loby Matise.								

To move among cells in the table, use the Up, Down, Left, and Right arrow keys. To move forward from one cell to the next, use the **Tab** key. Pressing the **Tab** key in the last cell of a row creates a new row.

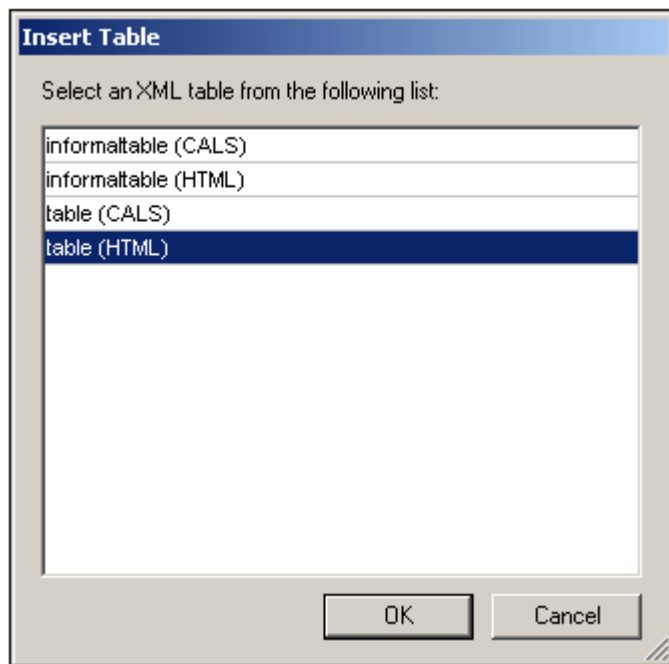
CALS/HTML Tables

CALS/HTML tables can be inserted by you, the user of Authentic View, for certain XML data structures that have been specified to show a table format. There are three steps involved when working with CALS/HTML tables: inserting the table; formatting it; and entering data. The commands for working with CALS/HTML tables are available as icons in the toolbar (see [CALS/HTML table editing icons](#)).

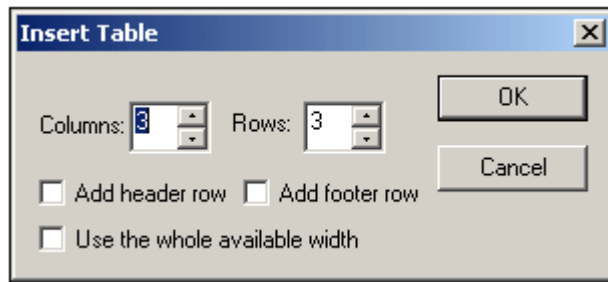
Inserting tables

To insert a CALS/HTML table do the following:

1. Place your cursor where you wish to insert the table, and click the  icon. (Note that where you can insert tables is determined by the schema.) The Insert Table dialog (*screenshot below*) appears. This dialog lists all the XML element data-structures for which a table structure has been defined. For example, in the screenshot below, the `informaltable` element and `table` element have each been defined as both a CALS table as well as an HTML table.



2. Select the entry containing the element and table model you wish to insert, and click **OK**.
3. In the next dialog (*screenshot below*), select the number of columns and rows, and specify whether a header and/or footer is to be added to the table and whether the table is to extend over the entire available width. Click **OK** when done.



For the specifications given in the dialog box shown above, the following table is created.

By using the **Table** menu commands, you can add and delete columns, and create row and column joins and splits. But to start with, you must create the broad structure.

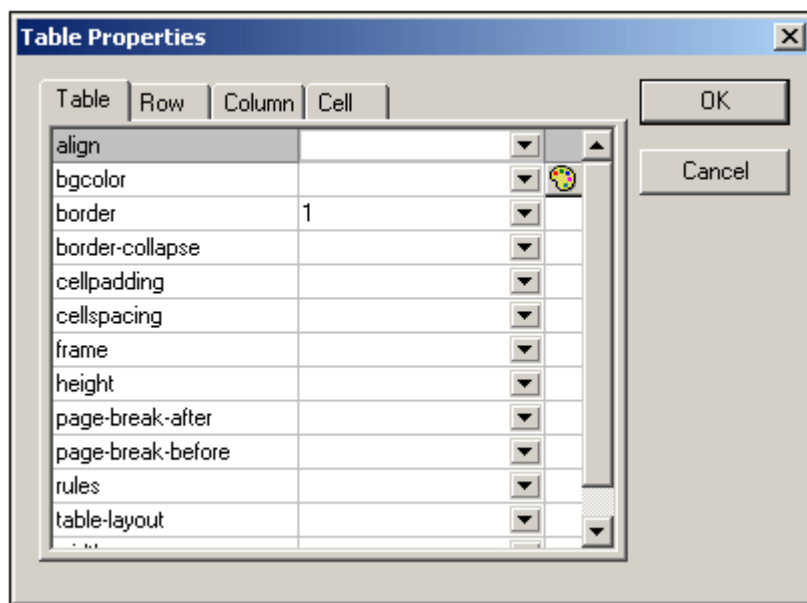
Formatting tables and entering data

The table formatting will already have been assigned in the document design. However, you might, under certain circumstances, be able to modify the table formatting. These circumstances are as follows:

- The elements corresponding to the various table structure elements must have the relevant CALS or HTML table properties defined as attributes (in the underlying XML Schema). Only those attributes that are defined will be available for formatting. If, in the design, values have been set for these attributes, then you can override these values in Authentic View.
- In the design, no `style` attribute containing CSS styles must have been set. If a `style` attribute containing CSS styles has been specified for an element, the `style` attribute has precedence over any other formatting attribute set on that element. As a result, any formatting specified in Authentic View will be overridden.

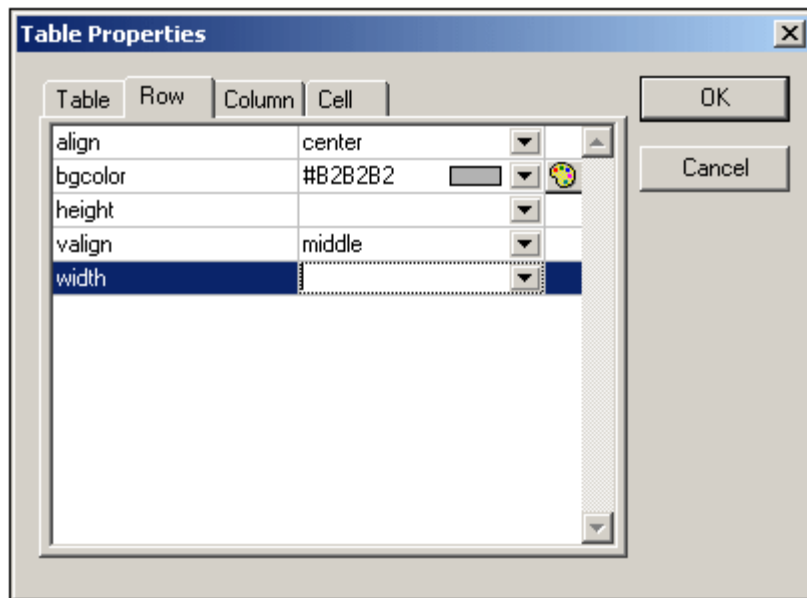
To format a table, row, column, or cell, do the following:

1. Place the cursor anywhere in the table and click the  (Table Properties) icon. This opens the Table Properties dialog (see *screenshot*), where you specify formatting for the table, or for a row, column, or cell.



2. Set the cellspacing and cellpadding properties to "0". Your table will now look like this:

3. Place the cursor in the first row to format it, and click the  (Table Properties) icon. Click the **Row** tab.



Since the first row will be the header row, set a background color to differentiate this row from the other rows. Note the Row properties that have been set in the figure above. Then enter the column header text. Your table will now look like this:

Name	Telephone	Email

Notice that the alignment is centered as specified.

- Now, say you want to divide the "Telephone" column into the sub-columns "Office" and "Home", in which case you would need to split the horizontal width of the Telephone column into two columns. First, however, we will split the vertical extent of the header cell to make a sub-header row. Place the cursor in the "Telephone" cell, and click the



(Split vertically) icon. Your table will look like this:

Name	Telephone	Email

- Now place the cursor in the cell below the cell containing "Telephone", and click the



(Split horizontally) icon. Then type in the column headers "Office" and "Home". Your table will now look like this:

Name	Telephone		Email
	Office	Home	

Now you will have to split the horizontal width of each cell in the "Telephone" column.

You can also add and delete columns and rows, and vertically align cell content, using the table-editing icons. The CALS/HTML table editing icons are described in the section titled, [CALS/HTML Table Editing Icons](#).

Moving among cells in the table

To move among cells in the CALS/HTML table, use the Up, Down, Right, and Left arrow keys.

Entering data in a cell

To enter data in a cell, place the cursor in the cell, and type in the data.

Formatting text

Text in a CALS/HTML table, as with other text in the XML document, must be formatted using XML elements or attributes. To add an element, highlight the text and double-click the required element in the Elements Entry Helper. To specify an attribute value, place the cursor within the text fragment and enter the required attribute value in the Attributes Entry Helper. After formatting the header text bold, your table will look like this.

Name	Telephone		Email
	Office	Home	

The text above was formatted by highlighting the text, and double-clicking the element `strong`, for which a global template exists that specifies bold as the font-weight. The text formatting becomes immediately visible.

Please note: For text formatting to be displayed in Authentic View, a global template with the required text formatting must have been created in StyleVision for the element in question.

CALS/HTML Table Editing Icons

The commands required to edit CALS/HTML tables are available as icons in the toolbar, and are listed below. Note that no corresponding menu commands exist for these icons.

For a full description of when and how CALS/HTML Tables are to be used, see [CALS/HTML Tables](#).

Insert table



The "Insert Table" command inserts a **CALS/HTML table** at the current cursor position.

Delete table



The "Delete table" command deletes the currently active table.

Append row



The "Append row" command appends a row to the end of the currently active table.

Append column



The "Append column" command appends a column to the end of the currently active table.

Insert row



The "Insert row" command inserts a row above the current cursor position in the currently active table.

Insert column



The "Insert column" command inserts a column to the left of the current cursor position in the currently active table.

Join cell left



The "Join cell left" command joins the current cell (current cursor position) with the cell to the left. The tags of both cells remain in the new cell, the column headers remain

unchanged and are concatenated.

Join cell right



The "Join cell right" command joins the current cell (current cursor position) with the cell to the right. The contents of both cells are concatenated in the new cell.

Join cell below



The "Join cell below" command joins the current cell (current cursor position) with the cell below. The contents of both cells are concatenated in the new cell.

Join cell above



The "Join cell above" command joins the current cell (current cursor position) with the cell above. The contents of both cells are concatenated in the new cell.

Split cell horizontally



The "Split cell Horizontally" command creates a new cell to the right of the currently active cell. The size of both cells, is now the same as the original cell.

Split cell vertically



The "Split cell Vertically" command creates a new cell below the currently active cell.

Align top



This command aligns the cell contents to the top of the cell.

Center vertically



This command centers the cell contents.

Align bottom

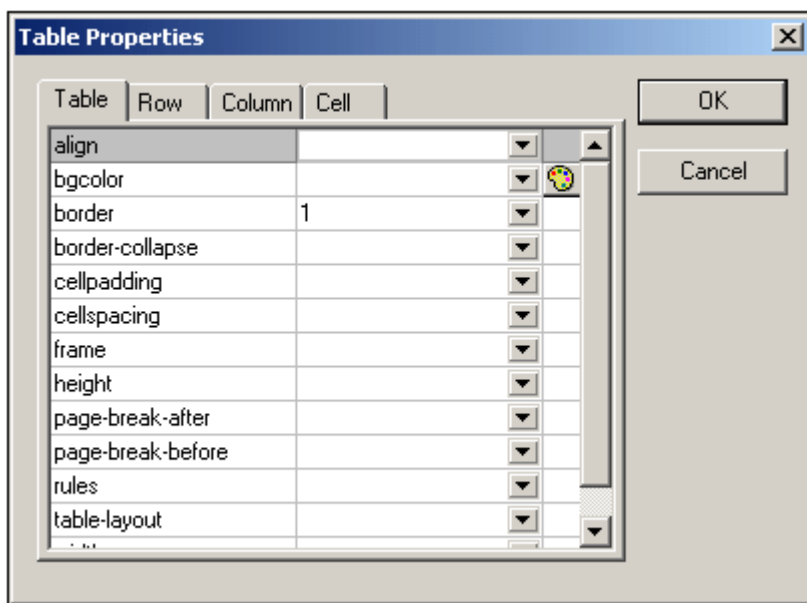


This command aligns the cell contents to the bottom of the cell.

Table properties



The "Table properties" command opens the Table Properties dialog box. This icon is only made active for HTML tables, it cannot be clicked for CALS tables.



8.2.3 Editing a DB

In Authentic View, you can edit database (DB) tables and save data back to a DB. This section contains a full description of interface features available to you when editing a DB table. The following general points need to be noted:

- The number of records in a DB table that are displayed in Authentic View may have been deliberately restricted by the designer of the StyleVision Power Stylesheet in order to make the design more compact. In such cases, only that limited number of records is initially loaded into Authentic View. Using the DB table row navigation icons (see [Navigating a DB Table](#)), you can load and display the other records in the DB table.
- You can [query the DB](#) to display certain records.
- You can add, modify, and delete DB records, and save your changes back to the DB. See [Modifying a DB Table](#).

To open a DB-based StyleVision Power Stylesheet in Authentic View:

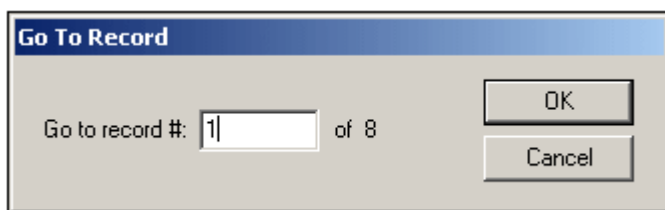
- Click **Authentic | Edit Database Data**, and browse for the required StyleVision Power Stylesheet.

Navigating a DB Table

The commands to navigate DB table rows are available as buttons in the Authentic View document. Typically, one navigation panel with either four or five buttons accompanies each DB table.



The arrow icons are, from left to right, Go to First Record in the DB Table; Go to Previous Record; Open the Go to Record dialog (see [screenshot](#)); Go to Next Record; and Go to Last Record.



To navigate a DB table, click the required button.

XML Databases

In the case of XML DBs, such as IBM DB2, one cell (or row) contains a single XML document, and therefore a single row is loaded into Authentic View at a time. To load an XML document that is in another row, use the [Authentic | Select New Row with XML Data for Editing](#) menu command.

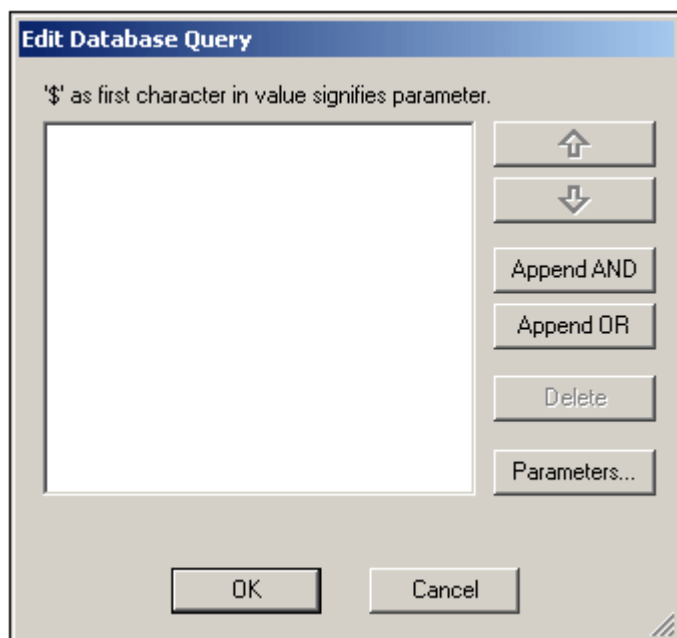
DB Queries

A DB query enables you to query the records of a table displayed in Authentic View. A query is made for an individual table, and only one query can be made for each table. You can make a query at any time while editing. If you have unsaved changes in your Authentic View document at the time you submit the query, you will be prompted about whether you wish to save **all** changes made in the document or discard **all** changes. Note that even changes made in other tables will be saved/discarded. After you submit the query, the table is reloaded using the query conditions.

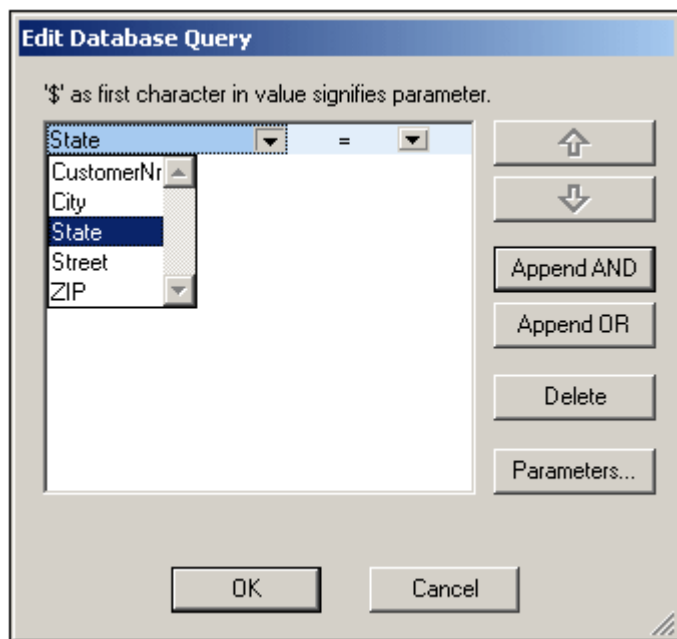
Please note: If you get a message saying that too many tables are open, then you can reduce the number of tables that are open by using a query to filter out some tables.

To create and submit a query:

1. Click the Query button  for the required table in order to open the Edit Database Query dialog (see *screenshot*). This button typically appears at the top of each DB table or below it. If a Query button is not present for any table, the designer of the StyleVision Power Stylesheet has not enabled the DB Query feature for that table.



2. Click the **Append AND** or **Append OR** button. This appends an empty criterion for the query (shown below).



4. Enter the expression for the criterion. An expression consists of: (i) a field name (available from the associated combo-box); (ii) an operator (available from the associated combo-box); and (iii) a value (to be entered directly). For details of how to construct expressions see the [Expressions in criteria](#) section.
5. If you wish to add another criterion, click the **Append AND** or **Append OR** button according to which logical operator (AND or OR) you wish to use to join the two criteria. Then add the new criterion. For details about the logical operators, see the section [Re-ordering criteria in DB Queries](#).

Expressions in criteria

Expressions in DB Query criteria consist of a field name, an operator, and a value. The **available field names** are the child elements of the selected top-level data table; the names of these fields are listed in a combo-box (see *screenshot above*). The **operators** you can use are listed below:

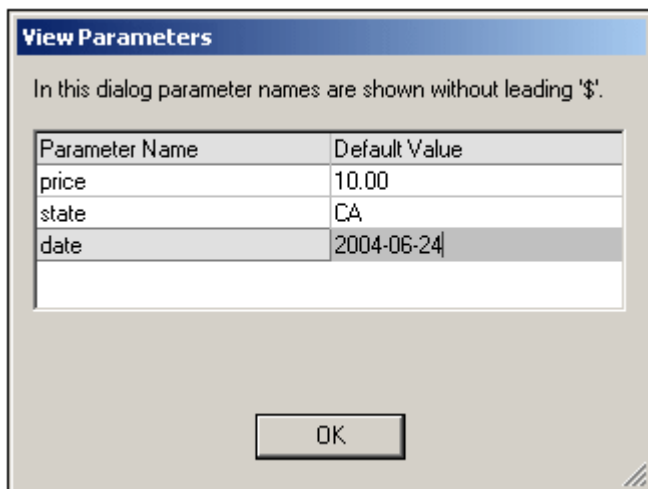
=	Equal to
<>	Not equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
LIKE	Phonetically alike
NOT LIKE	Phonetically not alike
IS NULL	Is empty
NOT NULL	Is not empty

If IS NULL or NOT NULL is selected, the Value field is disabled. **Values** must be entered without quotes (or any other delimiter). Values must also have the same formatting as that of the corresponding DB field; otherwise the expression will evaluate to `FALSE`. For example, if a criterion for a field of the `date` datatype in an MS Access DB has an expression `StartDate=25/05/2004`, the expression will evaluate to `FALSE` because the `date` datatype in an MS Access DB has a format of `YYYY-MM-DD`.

Using parameters with DB Queries

You can enter the name of a **parameter** as the value of an expression when creating queries. Parameters are variables that can be used instead of literal values in queries. When you enter it in an expression, its value is used in the expression. Parameters that are available have been defined by the SPS designer in the SPS and can be viewed in the View Parameters dialog (see *screenshot below*). Parameters have been assigned a default value in the SPS, which can be overridden by passing a value to the parameter via the command line (if and when the output document is compiled via the command line).

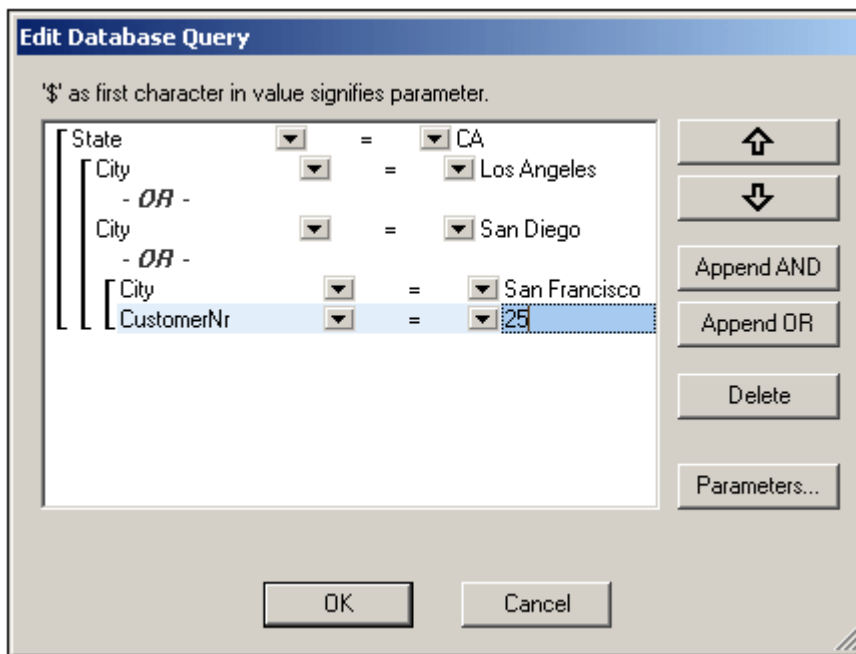
To view the parameters defined for the SPS, click the **Parameters** button in the Edit Database Query dialog. This opens the **View Parameters** dialog (see *screenshot*).



The View Parameters dialog contains **all** the parameters that have been defined for the stylesheet in the SPS and parameters must be edited in th stylesheet design.

Re-ordering criteria in DB Queries

The logical structure of the DB Query and the relationship between any two criteria or sets of criteria is indicated graphically. Each level of the logical structure is indicated by a square bracket. Two adjacent criteria or sets of criteria indicate the AND operator, whereas if two criteria are separated by the word **OR** then the OR operator is indicated. The criteria are also appropriately indented to provide a clear overview of the logical structure of the DB Query.



The DB Query shown in the screenshot above may be represented in text as:

```
State=CA AND ( City=Los Angeles OR City=San Diego OR ( City=San
Francisco AND CustomerNr=25) )
```

You can re-order the DB Query by moving a criterion or set of criteria up or down relative to the other criteria in the DB Query. To move a criterion or set of criteria, do the following:

1. Select the criterion by clicking on it, or select an entire level by clicking on the bracket that represents that level.
2. Click the Up or Down arrow button in the dialog.

The following points should be noted:

- If the adjacent criterion in the direction of movement is at the same level, the two criteria exchange places.
- A set of criteria (i.e. criterion within a bracket) changes position within the same level; it does not change levels.
- An individual criterion changes position within the same level. If the adjacent criterion is further outward/inward (i.e. not on the same level), then the selected criterion will move outward/inward, **one level at a time**.

To delete a criterion in a DB Query, select the criterion and click **Delete**.

Modifying a DB Query

To modify a DB Query:

1. Click the Query button . The Edit Database Query dialog box opens. You can now edit the expressions in any of the listed criteria, add new criteria, re-order criteria, or delete criteria in the DB Query.
2. Click **OK**. The data from the DB is automatically re-loaded into Authentic View so as to reflect the modifications to the DB Query.

Modifying a DB Table

Adding a record

To add a record to a DB table:

1. Place the cursor in the DB table row and click the  icon (to append a row) or the  icon (to insert a row). This creates a new record in the temporary XML file.
2. Click the **File | Save** command to add the new record in the DB. In Authentic View a row for the new record is appended to the DB table display. The `AltovaRowStatus` for this record is set to `A` (for Added).

When you enter data for the new record it is entered in bold and is underlined. This enables you to differentiate added records from existing records—if existing records have not been formatted with these text formatting properties. Datatype errors are flagged by being displayed in red.

The new record is added to the DB when you click **File | Save**. After a new record is saved to the DB, its `AltovaRowStatus` field is initialized (indicated with `---`) and the record is displayed in Authentic View as a regular record.

Modifying a record

To modify a record, place the cursor at the required point in the DB table and edit the record as required. If the number of displayed records is limited, you may need to navigate to the required record (see [Navigating a DB Table](#)).

When you modify a record, entries in all fields of the record are underlined and the `AltovaRowStatus` of all primary instances of this record is set to `U` (for Updated). All secondary instances of this record have their `AltovaRowStatus` set to `u` (lowercase). Primary and secondary instances of a record are defined by the structure of the DB—and correspondingly of the XML Schema generated from it. For example, if an Address table is included in a Customer table, then the Address table can occur in the Design Document in two types of instantiations: as the Address table itself and within instantiations of the Customer table. Whichever of these two types is modified is the type that has been primarily modified. Other types—there may be more than one other type—are secondary types. Datatype errors are flagged by being displayed in red.

The modifications are saved to the DB by clicking **File | Save**. After a modified record is saved to the DB, its `AltovaRowStatus` field is initialized (indicated with `---`) and the record is displayed in Authentic View as a regular record.

Please note:

- If even a single field of a record is modified in Authentic View, the entire record is updated when the data is saved to the DB.
- The date value `0001-01-01` is defined as a `NULL` value for some DBs, and could result in an error message.

Deleting a record

To delete a record:

1. Place the cursor in the row representing the record to be deleted and click the  icon. The record to be deleted is marked with a strikethrough. The `AltovaRowStatus` is set as follows: primary instances of the record are set to `D`; secondary instances to `d`; and records indirectly deleted to `X`. Indirectly deleted records are fields in the deleted record that are held in a separate table. For example, an Address table might be included in a Customer table. If a Customer record were to be deleted, then its corresponding Address record would be indirectly deleted. If an Address record in the Customer table were deleted, then the Address record in the Customer table would be primarily deleted, but the same record would be secondarily deleted in an independent Address table if this were instantiated.
2. Click **File | Save** to save the modifications to the DB.

Please note: Saving data to the DB resets the Undo command, so you cannot undo actions that were carried out prior to the save.

8.2.4 Working with Dates

There are two ways in which dates can be edited in Authentic View:

- Dates are entered or modified using the [Date Picker](#).
- Dates are entered or modified by [typing in the value](#).

The method the Authentic View user will use is defined in the SPS. Both methods are described in the two sub-sections of this section.

Note on date formats

In the XML document, dates can be stored in one of several date datatypes. Each of these datatypes requires that the date be stored in a particular lexical format in order for the XML document to be valid. For example, the `xs:date` datatype requires a lexical format of `YYYY-MM-DD`. If the date in an `xs:date` node is entered in anything other than this format, then the XML document will be invalid.

In order to ensure that the date is entered in the correct format, the SPS designer can include the graphical Date Picker in the design. This would ensure that the date selected in the Date Picker is entered in the correct lexical format. If there is no Date Picker, the Authentic View should take care to enter the date in the correct lexical format. Validating the XML document could provide useful tips about the required lexical format.

Date Picker

The Date Picker is a graphical calendar used to enter dates in a standard format into the XML document. Having a standard format is important for the processing of data in the document. The Date Picker icon appears near the date field it modifies (see *screenshot*).



To display the Date Picker (see *screenshot*), click the Date Picker icon.

Location of logo: nanonull.gif

Last Updated: 2003-09-01

Nanonull, Inc.

Location: US

September 2003

M	T	W	T	F	S	S
25	26	27	28	29	30	31
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	1	2	3	4	5

Today No Timezone

To select a date, click on the desired date, month, or year. The date is entered in the XML document, and the date in the display is modified accordingly. You can also enter a time zone if this is required.

Text Entry

For date fields that do not have a Date Picker (see *screenshot*), you can edit the date directly by typing in the new value.

Please note: When editing a date, you must not change its format.

Invoice Number: 001
2006-03-10
Customer: The ABC Company
Invoice Amount: 40.00

If you edit a date and change it such that it is out of the valid range for dates, the date turns red to alert you to the error. If you place the mouse cursor over the invalid date, an error message appears (see *screenshot*).

Invoice Number: 001
2006-03-32
Customer: ERROR: Invalid value for datatype date in element
Invoice Amount: 40.00

If you try to change the format of the date, the date turns red to alert you to the error (see *screenshot*).

Invoice Number: 001
2006/03/10
Customer: The ABC Company
Invoice Amount: 40.00

8.2.5 Defining Entities

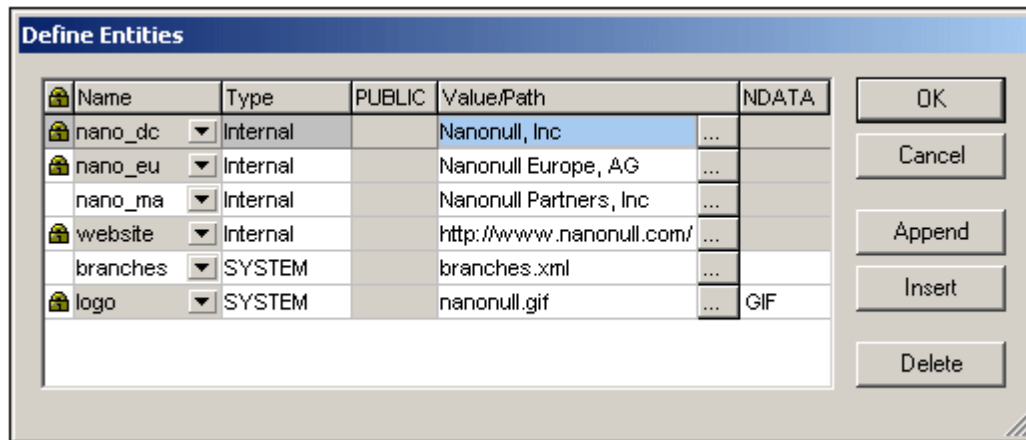
You can define entities for use in Authentic View, whether your document is based on a DTD or an XML Schema. Once defined, these entities are displayed in the Entities Entry Helper and in the **Insert Entity** submenu of the context menu. When you double-click on an entity in the Entities Entry Helper, that entity is inserted at the cursor insertion point.

An entity is useful if you will be using a text string, XML fragment, or some other external resource in multiple locations in your document. You define the entity, which is basically a short name that stands in for the required data, in the Define Entities dialog. After defining an entity you can use it at multiple locations in your document. This helps you save time and greatly enhances maintenance.

There are two broad types of entities you can use in your document: a **parsed entity**, which is XML data (either a text string or a fragment of an XML document), or an **unparsed entity**, which is non-XML data such as a binary file (usually a graphic, sound, or multimedia object). Each entity has a name and a value. In the case of parsed entities the entity is a placeholder for the XML data. The value of the entity is either the XML data itself or a URI that points to a .xml file that contains the XML data. In the case of unparsed entities, the value of the entity is a URI that points to the non-XML data file.

To define an entity:

1. Click **Authentic | Define XML Entities....** This opens the Define Entities dialog (*screenshot below*).



2. Enter the name of your entity in the Name field. This is the name that will appear in the Entities Entry Helper.
3. Enter the type of entity from the drop-down list in the Type field. The following types are possible: An **Internal** entity is one for which the text to be used is stored in the XML document itself. Selecting **PUBLIC** or **SYSTEM** specifies that the resource is located outside the XML file, and will be located with the use of a public identifier or a system identifier, respectively. A system identifier is a URI that gives the location of the resource. A public identifier is a location-independent identifier, which enables some processors to identify the resource. If you specify both a public and system identifier, the public identifier resolves to the system identifier, and the system identifier is used.
4. If you have selected PUBLIC as the Type, enter the public identifier of your resource in the PUBLIC field. If you have selected Internal or SYSTEM as your Type, the PUBLIC field is disabled.
5. In the Value/Path field, you can enter any one of the following:
 - If the entity type is Internal, enter the text string you want as the value of your entity. Do not enter quotes to delimit the entry. Any quotes that you enter will be treated as part of the text string.
 - If the entity type is SYSTEM, enter the URI of the resource or select a resource on your local network by using the Browse button. If the resource contains parsed data, it must be an XML file (i.e., it must have a .xml extension). Alternatively, the resource can be a binary file, such as a GIF file.
 - If the entity type is PUBLIC, you must additionally enter a system identifier in this field.

6. The NDATA entry tells the processor that this entity is not to be parsed but to be sent to the appropriate processor. The NDATA field must therefore contain some value to indicate that the entity is an unparsed entity.

Dialog features

You can do the following in the Define Entities dialog:

- Append entities
- Insert entities
- Delete entities
- Sort entities by the alphabetical value of any column by clicking the column header; clicking once sorts in ascending order, twice in descending order.
- Resize the dialog box and the width of columns.
- Locking. Once an entity is used in the XML document, it is locked and cannot be edited in the Define Entities dialog. Locked entities are indicated by a lock symbol in the first column. Locking an entity ensures that the XML document valid with respect to entities. (The document would be invalid if an entity is referenced but not defined.)
- Duplicate entities are flagged.

Limitations of entities

- An entity contained within another entity is not resolved, either in the dialog, Authentic View, or XSLT output, and the ampersand character of such an entity is displayed in its escaped form, i.e. `&`.
- External unparsed entities that are not image files are not resolved in Authentic View. If an image in the design is defined to read an external unparsed entity and has its URI set to be an entity name (for example: ' `logo` '), then this entity name can be defined in the Define Entities dialog (see *screenshot above*) as an external unparsed entity with a value that resolves to the URI of the image file (as has been done for the `logo` entity in the screenshot above).

8.2.6 XML Signatures

An SPS can be designed with an XML signature configured for Authentic View. When XML signatures are enabled in the SPS, the Authentic View user can digitally sign the Authentic XML file with the enabled signature. After the document has been signed, any modification to it will cause the verification of the signature to fail. Whenever a signed Authentic XML document is opened in the Authentic View of any Altova product, the verification process will be run on the document and the result of the verification will be displayed in a window.

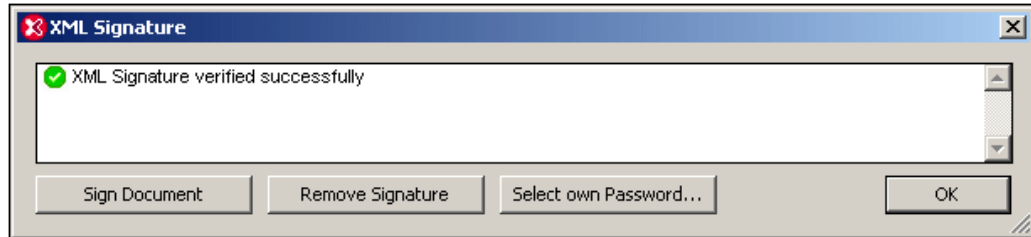
Note: XML signatures can be used, and will be verified, in the Authentic View of Enterprise and Professional editions of the following Altova products: Authentic Desktop, Authentic Browser, XMLSpy, and StyleVision.

XML signature actions

The following Authentic View user actions for signatures are possible:

- *Choosing the certificate/password:* Signatures are authenticated with either a certificate or a password. The authentication object (certificate or password) is required when the signature is created and again when it is verified. If an Authentic XML document has a signature-enabled SPS assigned to it, the SPS might specify a default certificate or password for the signature. Whether a default certificate or password has been specified or not, the signature can be configured to allow the Authentic View user to select an own certificate/password. The Authentic View user can do this at any time in the XML Signature dialog (*screenshot below*). Selecting an own certificate/password

overrides the default certificate/password. The own certificate/password is stored in memory and is used for the current session. If, after an own certificate/password has been selected, the Authentic View user closes the file or the application, the SPS reverts to its default setting for the certificate/password.



- *Signing the document:* The Authentic XML document can be signed either automatically or manually. Automatic signing will have been specified in the signature configuration by the SPS designer and causes the Authentic XML document to be signed automatically when it is saved. If the automatic-signing option has not been activated, the document can be signed manually. This is done by clicking the XML Signature toolbar icon  or the **Authentic | XML Signature** command, and, in the XML Signature dialog that then pops up (*screenshot above*), clicking the **Sign Document** button. Note that signing the document with an embedded signature would require the schema to allow the `Signature` element as the last child element of the root (document) element. Otherwise the document will be invalid against the schema. When signing the document, the authentication object and the placement of the signature are determined according to the signature configuration. You must ensure that you have access to the authentication information. For more information about this, consult your SPS designer.
- *Verifying the Authentic XML document:* If an SPS has XML Signatures enabled, the verification process will be run on the signature each time the Authentic View XML document is loaded. If the password or certificate key information is not saved with the SPS and signature, respectively, the Authentic View user will be prompted to enter the password or select a certificate for verification. Note that if an embedded signature is generated, it will be saved with the XML file when the XML file is saved. The generated signature must be explicitly removed (via the **Remove Signature** button of the XML Signature dialog; see *screenshot above*) if you do not wish to save it with the XML file. Similarly, if a detached signature is generated, it too must be explicitly removed if it is not required.

8.2.7 Images in Authentic View

Authentic View allows you to specify images that will be used in the final output document (HTML, RTF, PDF and Word 2007). You should note that some image formats might not be supported in some formats or by some applications. For example, the SVG format is supported in PDF, but not in RTF and would require a browser add-on for it to be viewed in HTML. So, when selecting an image format, be sure to select a format that is supported in the output formats of your document. Most image formats are supported across all the output formats (see *list below*).

Authentic View is based on Internet Explorer, and is able to display most of the image formats that your version of Internet Explorer can display. The following commonly used image formats are supported:

- GIF
- JPG
- PNG

- BMP
- WMF (Microsoft Windows Metafile)
- EMF (Enhanced Metafile)
- SVG (for PDF output only)

Relative paths

Relative paths are resolved relative to the SPS file.

8.2.8 Keystrokes in Authentic View

Enter key

In Authentic View the **Enter** key is used to append additional elements when it is in certain cursor locations. For example, if the chapter of a book may (according to the schema) contain several paragraphs, then pressing **Enter** inside the text of the paragraph causes a new paragraph to be appended immediately after the current paragraph. If a chapter can contain one title and several paragraphs, pressing **Enter** inside the chapter but outside any paragraph element (including within the title element) causes a new chapter to be appended after the current chapter (assuming that multiple chapters are allowed by the schema).

Please note: The **Enter** key does **not** insert a new line. This is the case even when the cursor is inside a text node, such as paragraph.

Using the keyboard

The keyboard can be used in the standard way, for typing and navigating. Note the following special points:

- The **Tab** key moves the cursor forward, stopping before and after nodes, and highlighting node contents; it steps over static content.
- The `add. . .` and `add Node` hyperlinks are considered node contents and are highlighted when tabbed. They can be activated by pressing either the spacebar or the **Enter** key.

8.3 Authentic Scripting

The **Authentic Scripting** feature provides more flexibility and interactivity to SPS designs. These designs can be created or edited in StyleVision Enterprise and Professional editions, and can be viewed in the Authentic View of the Enterprise and Professional editions of Altova products.

A complete listing of support for this feature in Altova products is given in the table below. Note, however, that in the trusted version of Authentic Browser plug-in, internal scripting is turned off because of security concerns.

Altova Product	Authentic Scripts Creation	Authentic Scripts Enabled
StyleVision Enterprise	Yes	Yes
StyleVision Professional	Yes	Yes
StyleVision Standard *	No	No
XMLSpy Enterprise	No	Yes
XMLSpy Professional	No	Yes
XMLSpy Standard	No	No
AuthenticDesktop Enterprise	No	Yes
AuthenticDesktop Community	No	No
Authentic Browser Plug-in Community **	No	No
Authentic Browser Plug-in Enterprise Trusted ***	No	Yes
Authentic Browser Plug-in Enterprise Untrusted	No	Yes

* *No AuthenticView*

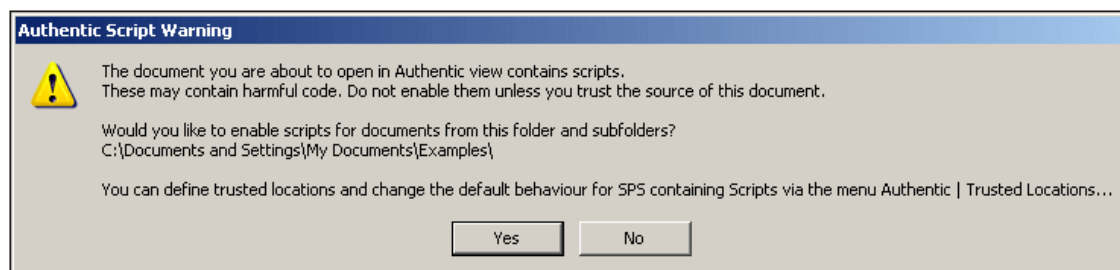
** *Both Trusted and Untrusted versions*

*** *Scripted designs displayed. No internal macro execution or event handling. External events fired.*

Authentic Scripts behave in the same way in all Altova products, so no product-specific code or settings are required.

Authentic Script Warning Dialog

If a PXF file, or an XML file linked to an SPS, contains a script and the file is opened or switched to Authentic View, then a warning dialog (*screenshot below*) pops up.



You can choose one of the following options:

- Click **Yes**, to add the folder containing the file to the Trusted Locations list for Authentic scripts. Subsequently, all files in the trusted folder will be opened In Authentic View without this warning dialog being displayed first. The Trusted Locations list can be accessed via the menu command [Authentic | Trusted Locations](#), and modified.
- Click **No** to not add the folder containing the file to the Trusted Locations list. The file will be displayed in Authentic View with scripts disabled. The Authentic Script Warning dialog will appear each time this file is opened in Authentic View. To add the file's folder to the Trusted Locations list subsequently, open the Trusted locations dialog via the menu command [Authentic | Trusted Locations](#), and add the folder or modify as required.

For a description of the Trusted Locations dialog, see the description of the [Authentic | Trusted Locations](#) menu command in the User Reference.

Note: When XMLSpy is accessed via its COM interface (see [Programmers' Reference](#) to see how this can be done), **the security check is not done** and the **Authentic Script Warning dialog is not displayed**.

How Authentic Scripting works

The designer of the SPS design can use Authentic Scripting in two ways to make Authentic documents interactive:

- By assigning scripts for user-defined actions (macros) to design elements, toolbar buttons, and context menu items.
- By adding to the design event handlers that react to Authentic View events.

All the scripting that is required for making Authentic documents interactive is done within the StyleVision GUI (Enterprise and Professional editions). Forms, macros and event handlers are created within the Scripting Editor interface of StyleVision and these scripts are saved with the SPS. Then, in the Design View of StyleVision, the saved scripts are assigned to design elements, toolbar buttons, and context menus. When an XML document based on the SPS is opened in an Altova product that supports Authentic Scripting (*see table above*), the document will have the additional flexibility and interactivity that has been created for it.

Documentation for Authentic Scripting

The documentation for Authentic Scripting is available in the documentation of StyleVision. It can be viewed online via the [Product Documentation page](#) of the [Altova website](#).

9 HTML, CSS, JSON

XMLSpy provides intelligent editing features for [HTML](#), [CSS](#), and [JSON](#) documents. All three types of documents can be edited in [Text View](#). Additionally, JSON documents can be edited in [Grid View](#), and the active HTML document can be previewed in Browser View.

The intelligent editing features of each type of document is described separately in the sub-sections of this section: [HTML](#), [CSS](#), and [JSON](#).

9.1 HTML

HTML documents can be edited in Text View, and the edited page can then be viewed immediately in Browser View. Text View provides a number of useful HTML editing features. These are described in detail in [Text View](#), but the main features, as well as HTML-specific options, are listed below.

Support level

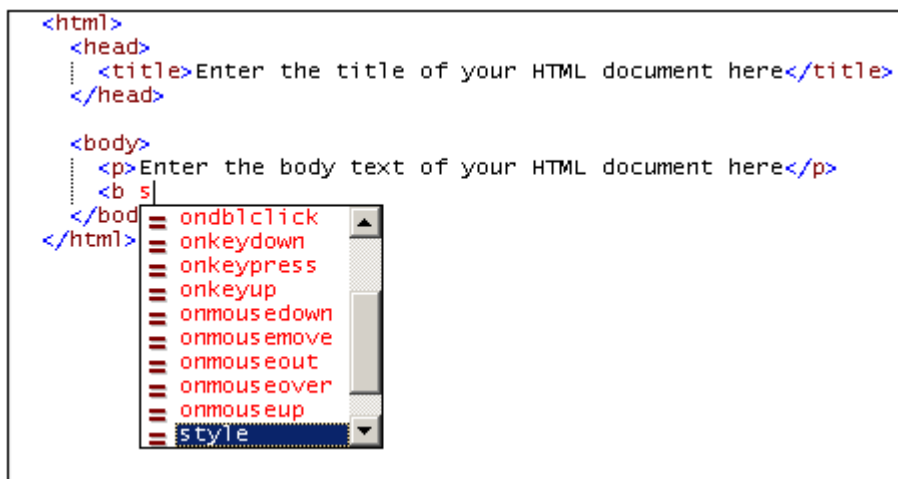
XMLSpy supports HTML 4.0 and HTML 5.0. Entry-helper and intelligent editing are available for the respective HTML versions. These features are described below.

Entry helpers

Elements, Attributes and Entities entry helpers are available when an HTML document is active. The entry helpers are context-sensitive; the items displayed in the entry helpers are those available at the current cursor location. Use the HTML entry helpers as described in [Text View](#).

Auto-completion

As you type markup text into your HTML document, XMLSpy provides Auto-completion help. A pop-up containing a list of all nodes available at the cursor insertion point is displayed. As you type, the selection jumps to the first closest match in the list (*see screenshot below*). Click the selected item to insert it at the cursor insertion point.



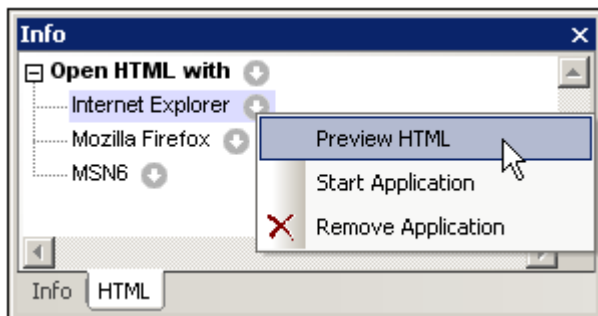
Auto-completion for elements appears when the left bracket of node tags is entered. When the start tag of an element node is entered in the document, the end tag is automatically inserted as well. This ensures well-formedness.

Auto-completion for attributes appears when a space is entered after the element name in a start tag. When you click an attribute name in the Auto-completion pop-up, the attribute is entered with quotes characters and the cursor positioned between the quotes.

The Entities entry helper contains character entities from the HTML 4.0 and HTML 5.0 entity sets, [Latin-1](#), [special characters](#), and [symbols](#).

HTML Info window

The HTML Info window (*screenshot below*) lists applications that can be used to quickly access the active HTML file. For example, if an HTML file is active in XMLSpy, double-clicking the Mozilla Firefox item in the HTML Info Window starts an instance of Mozilla Firefox and loads the active HTML document in it.



Note the following usage points:

- The icon to the right of the *Open HTML With* item enables applications to be added to the *Open HTML With* list. All the browsers installed on the system, or any other application (such as a text editor), can be added via the menu commands accessed via the *Open HTML With* icon. The associated applications would typically be browser or editor applications.
- After an application has been added to the *Open HTML With* list (except when added with the **Add Installed Browsers** command), its name in the *Open HTML With* list can be changed by selecting it, pressing **F2**, and editing the name.
- The icons to the right of each application listed in the *Open HTML With* list each opens a menu containing commands to: (i) open the application; (ii) open the application and load the linked HTML file; (iii) remove the application from the list. Double-clicking an application name opens the linked HTML file in that application.
- Applications added to or removed from the *Open HTML With* list are also added to or removed from the CSS Info window.

Assigning a DTD

For XHTML documents, a DTD or XML Schema can be assigned via the **DTD/Schema** menu, which enables you to browse for the required DTD or XML Schema file. An XHTML document can be [edited exactly like an XML document](#).

Browser View commands

Browser View commands are available in the **Browser** menu.

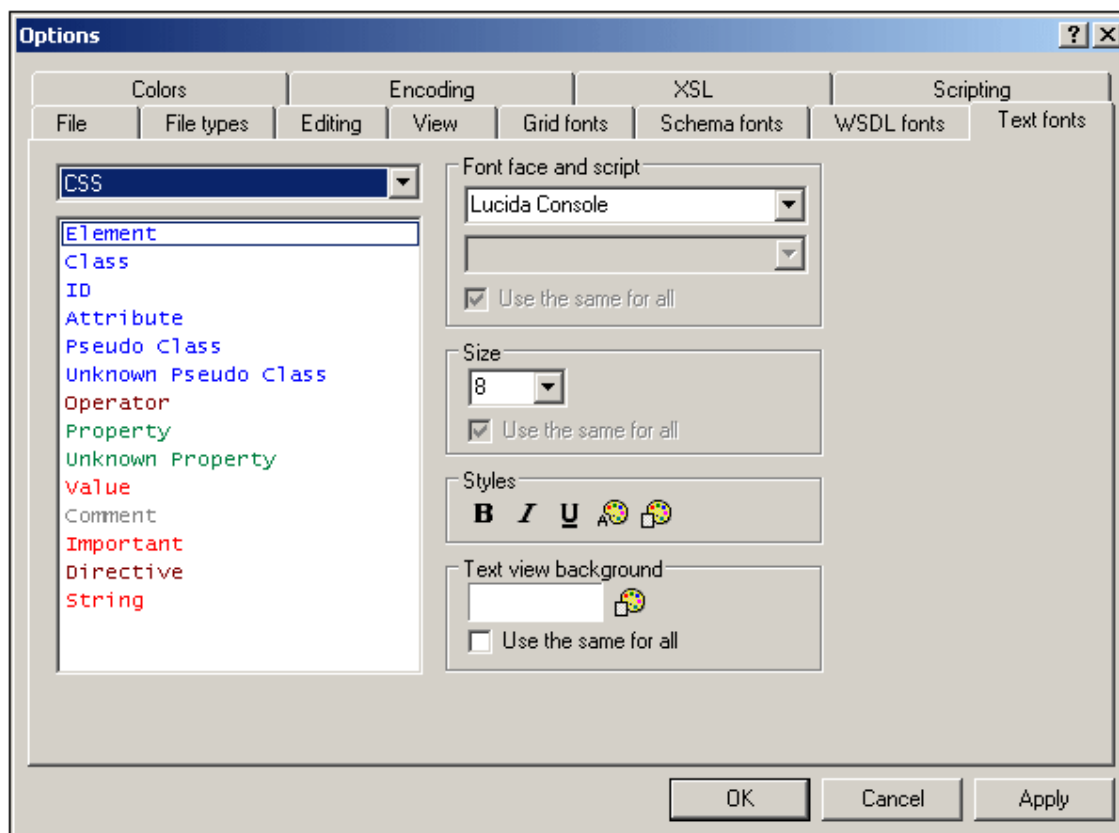
9.2 CSS

CSS documents can be edited using Text View's intelligent editing features. These features, as they apply to the editing of CSS documents, are listed below.

Syntax coloring: A CSS rule consists of a selector, one or more properties, and the values of those properties. These three components may be further sub-divided into more specific categories; for example, a selector may be a class, pseudo-class, ID, element, or attribute. Additionally, a CSS document can contain other items than rules: for example, comments. In Text View, each such category of items can be displayed in a different color (*screenshot below*) according to settings you make in the Options dialog (*see below*).

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

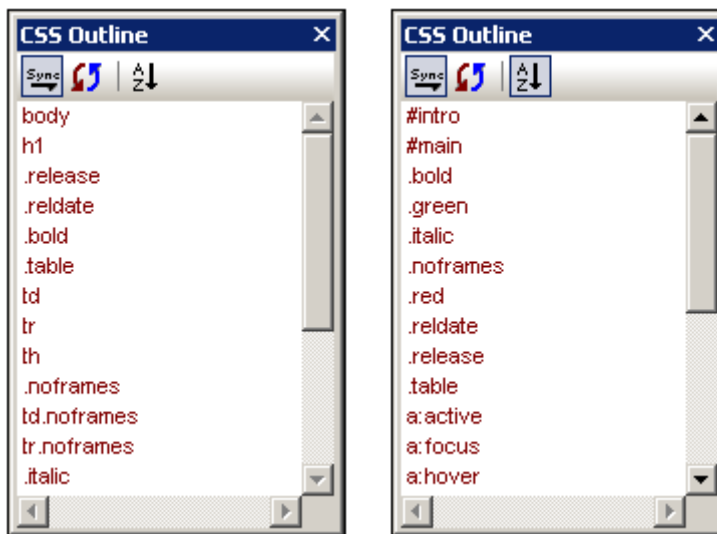
You can set the colors of the various CSS components in the Text Fonts tab of the Options dialog (*screenshot below*). In the combo box at top left, select CSS, and then select the required color (in the Styles pane) for each CSS item.



Folding margins: Each rule can be collapsed and expanded by clicking, respectively, the minus and plus icons to the left of the rule (*see screenshot above*). Note also that the pair of curly braces that delimit a rule (*screenshot above*) turns bold when the cursor is placed either before or after one of the curly braces. This indicates clearly where the definition of a particular rule starts and ends.

CSS outline: The CSS Outline entry helper (*screenshots below*) provides an outline of the document in terms of its selectors. Clicking a selector in the CSS Outline highlights it in the document. In the screenshot at left below, the selectors are unsorted and are listed in the order in which they appear in the document. In the screenshot at right, the Alphabetical Sorting feature has been toggled on (using the toolbar icon), and the selectors are sorted alphabetically.

You should note the following points: (i) For evaluating the alphabetical order of selectors, all parts of the selector are considered, including the period, hash, and colon characters; (ii) If the CSS document contains several selectors grouped together to define a single rule (e.g. `h4, h5, h6 { . . . }`), then each selector in the group is listed separately.



The icons in the toolbar of the CSS Outline entry helper, from left to right, do the following:



Toggles automatic synchronization (with the document) on and off. When auto-synchronization is switched on, selectors are entered in the entry helper even as you type them into the document.

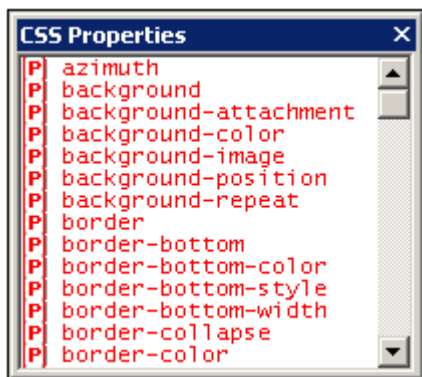


Synchronizes the entry helper with the current state of the document.

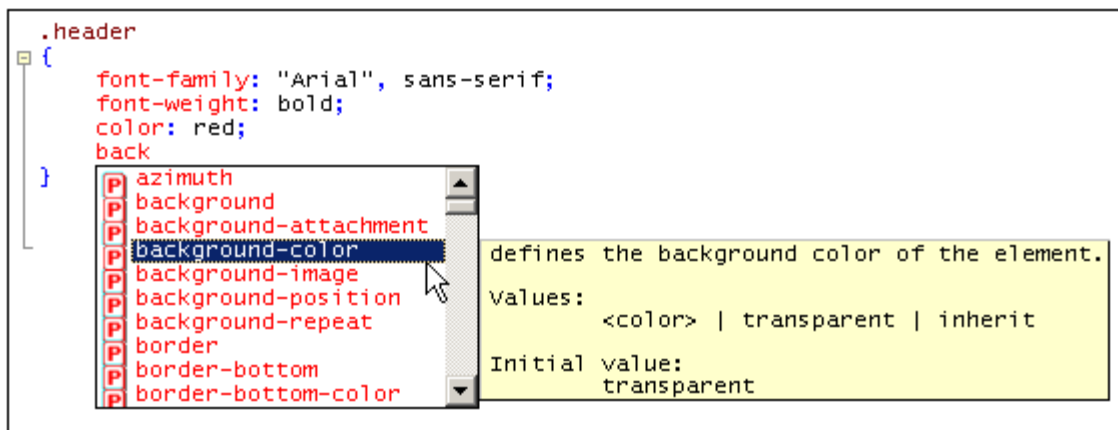


Toggles alphabetical sorting on and off. When off, the selectors are listed in the order in which they appear in the document. When sorted alphabetically, ID selectors appear first because they are prefaced by a hash (e.g. `#intro`).

Properties entry helper: The Properties entry helper (*screenshot below*) provides a list of all CSS properties, arranged alphabetically. A property can be inserted at the cursor insertion point by double-clicking the property.



Auto-completion of properties and tooltips for properties: As you start to type the name of a property, XMLSpy prompts you with a list of properties that begin with the letters you have typed (*screenshot below*). Alternatively, you can place the cursor anywhere inside a property name and then press **Ctrl+Space** to pop up the list of CSS properties.

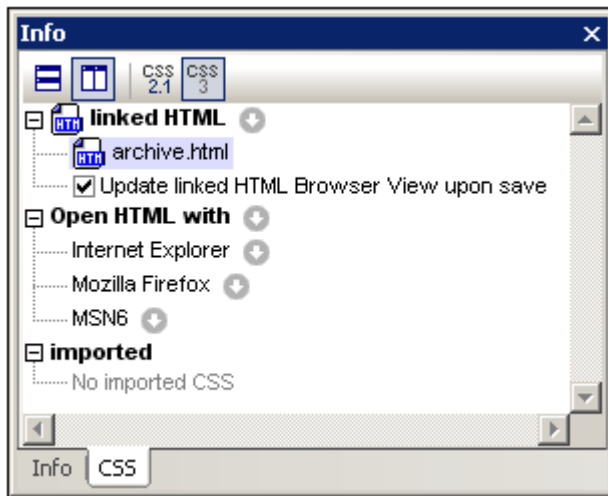


You can view a tooltip containing the definition of a property and its possible values by scrolling down the list or navigating the list with the Up and Down keys of your keyboard. The tooltip for the highlighted property is displayed. To insert a property, either press **Enter** when it is selected, or click it.

CSS Info window

When a CSS file is active, the CSS Info window (*screenshot below*) is enabled. The CSS Info window provides the following functionality:

- It enables you to switch between CSS 2.1 and CSS 3.0. The entry helpers and intelligent editing features of the GUI will be switched according to the CSS version selected in the toolbar of the Info window.
- It enables the CSS file to be linked to an HTML file. This functionality enables you to modify the CSS document and view the effect of changes immediately. Additionally, the linked HTML file can be opened in multiple browsers via the CSS Info window, thus enabling changes in the CSS document to be viewed in multiple browsers.
- The CSS Info window lists the imported CSS stylesheets, thus giving you an overview of the import structure of the active CSS stylesheet.



Note the following usage points:

- The toolbar of the Info window contains icons for CSS 2.1 and CSS 3.0. Select the version you want in order to switch entry helpers and intelligent editing features to the selected CSS version.
- Only one HTML file can be linked to the active CSS document. Do this by clicking the icon to the right of the *Linked HTML* item, then selecting the command **Set Link to HTML** and browsing for the required HTML file. The linked HTML file will be listed under the *Linked HTML* item in the Info window (see screenshot below). Creating this link does not modify the CSS document or the HTML document in any way. The link serves to set up an HTML file to which the active CSS document can be applied for testing.
- Double-clicking the Linked HTML file listing opens the HTML file in XMLSpy.
- The toolbar icons enable you to horizontally and vertically tile the CSS document and the HTML file.
- When changes to the CSS document are saved, the HTML file that is open in XMLSpy can be automatically updated. To enable these automatic updates, check the *Update Linked HTML Browser* check box. Note that these updates will only occur if the HTML file contains a reference to the CSS document being edited.
- To change the linked HTML file, select another HTML file via the **Set Link to HTML** command.
- To remove the link to the HTML file, click the icon to the right of the *Linked HTML* item and select the command **Remove Link**.
- The icon to the right of the *Open HTML With* item enables applications to be added to the *Open HTML With* list. All the browsers installed on the system, or any other application (such as a text editor), can be added via the menu commands accessed via the *Open HTML With* icon. The associated applications would typically be browser or editor applications.
- After an application has been added to the *Open HTML With* list (except when added with the **Add Installed Browsers** command), its name in the *Open HTML With* list can be changed by selecting it, pressing **F2**, and editing the name.
- The icons to the right of each application listed in the *Open HTML With* list each opens a menu containing commands to: (i) open the application; (ii) open the application and load the linked HTML file; (iii) remove the application from the list. Double-clicking an application name opens the linked HTML file in that application.
- Applications added to or removed from the *Open HTML With* list are also added to or removed from the HTML Info window.
- The *Imported* item displays a list of the CSS files imported by the active CSS document.

9.3 JSON

Altova website:  [JSON Editor](#)

JSON documents can be edited using Grid View and the intelligent editing features of Text View.

- [Grid View](#) provides a number of editing features that are very useful for editing large files; these features are described in the [Grid View section](#).
- The [Text View](#) features, as they apply to the editing of JSON documents, are listed below.

Note: XMLSpy also provides conversion between JSON and XML in both directions. Clicking the [Convert from/to JSON](#) command in XMLSpy converts the active document to the other format.

Folding margins: Each JSON element can be collapsed and expanded by clicking, respectively, the minus and plus icons to the left of the rule (see *screenshot below*). Note also that line-numbering is enabled.

```

10  "OrgChart": {
11      "xmlns": "http://www.xmlspy.com/schemas/orgchart",
12      "xmlns:ipo": "http://www.altova.com/IPO",
13      "xmlns:xsi": "http://www.w3.org/2001/XMLSchema-instance",
14      "xmlns:ts": "http://www.xmlspy.com/schemas/textstate",
15      "xsi:schemaLocation": "http://www.xmlspy.com/schemas/orgchart OrgChart.xsd",
16      "CompanyLogo": {
17          "href": "nanonull.gif"
18      },

```

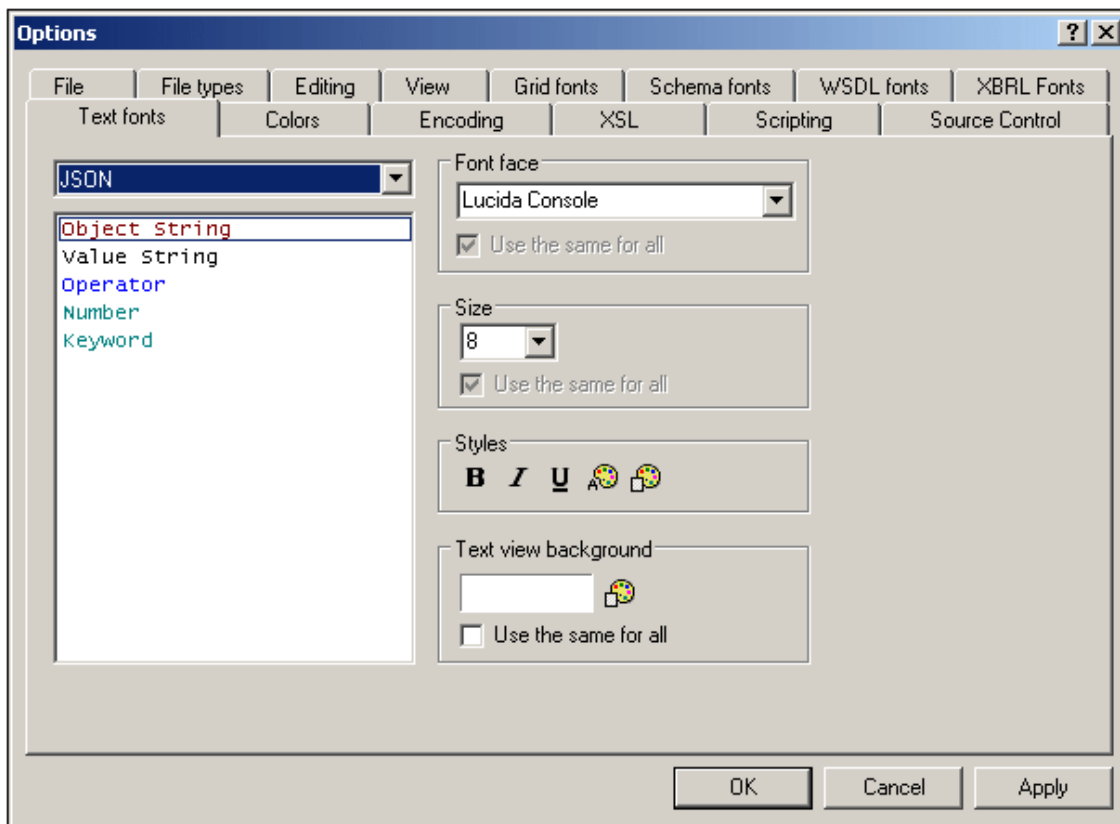
Structural marking: The pair of curly braces or square brackets that delimit a JSON element (*screenshot below*) turns bold when the cursor is placed either before or after one of the braces or brackets. This indicates where the definition of a particular element starts and ends.

```

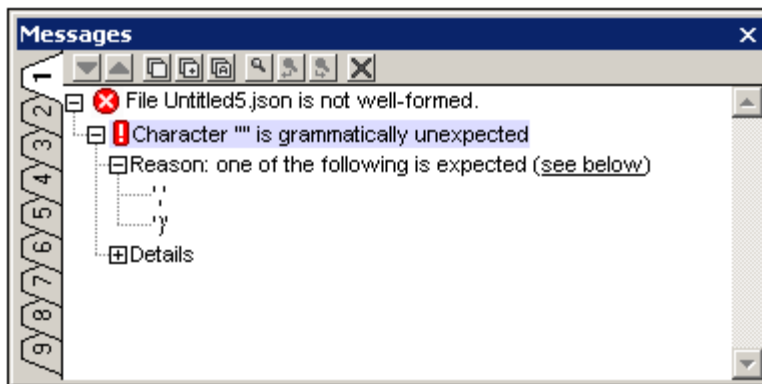
"Department": [ { "Name": "Administration",
"Person": [ { "First": "Vernon",
"Last": "Callaby",
"Title": "Office Manager",
"PhoneExt": 582,
"EMail": "v.callaby@nanonull.com",
"Shares": 1500,
"LeaveTotal": 25,
"LeaveUsed": 4,
"LeaveLeft": 21}],

```

Syntax coloring: A JSON document is made up of object strings, value strings, operators, numbers and keywords. In Text View, each category of items can be displayed in a different color (*screenshot above*) according to settings you make in the [Options dialog](#) (*screenshot below*). You can set the colors of the various JSON components in the Text Fonts tab of the Options dialog (*screenshot below*). In the combo box at top left, select JSON, and then select the required color (in the Styles pane) for each JSON item.



Syntax checking: The syntax of a JSON document can be checked by selecting the command **XML | Check Well-Formedness (F7)**. The results of the well-formed check are displayed in the Messages window (screenshot below).



The error message in the screenshot above points out an error in the document: Two double quotes (a closing quote followed by an opening quote) occur consecutively; syntactically a comma or a closing curly brace is required between the two quotes.

Converting between XML and JSON

The **Convert to/from JSON** command in the **Convert** menu converts an XML document, if it is the active document, to a JSON document. Also, if a JSON document is active, it can be converted to an XML document. For more information, see the [User Reference](#).

10 WSDL and SOAP

Altova website:  [WSDL Editor](#)

This section describes XMLSpy's WSDL and SOAP functionality.

WSDL

A WSDL document is an XML document that describes a web service. XMLSpy supports WSDL 1.1 and WSDL 2.0. You can create and edit both WSDL 1.1 and WSDL 2.0 documents in XMLSpy's WSDL View, which automatically provides the correct editing environment for whichever WSDL version is being edited.

In XMLSpy's WSDL View a WSDL document can be constructed using graphical building blocks, thus greatly simplifying their creation. [WSDL View](#) is described in the section, [Editing Views](#). For a hands-on description of creating a WSDL document, see the [WSDL Tutorial](#) in this documentation. You can also view and edit WSDL documents in [Text View](#) and [Grid View](#). In these two views, WSDL documents are edited as straightforward [XML documents](#).

[XML signatures](#) for WSDL files in WSDL View can be created as external files and can be "enveloped" in the WSDL file. How to work with signatures is described in the section, [XML Signatures](#).

SOAP

SOAP is an XML messaging specification, and it is used to transmit messages between applications. In XMLSpy, not only can you create and edit a SOAP document in [Text View](#) and [Grid View](#) with XMLSpy's intelligent editing features for [XML documents](#), but you can generate a SOAP request file from a WSDL file. How to generate a SOAP request from a WSDL file is described in the [WSDL Tutorial](#). XMLSpy is able to also send and receive SOAP requests (using commands in the [SOAP menu](#)). Additionally, you can debug SOAP requests with XMLSpy's [SOAP Debugger](#), which is described in a sub-section of this section.

10.1 WSDL Tutorial

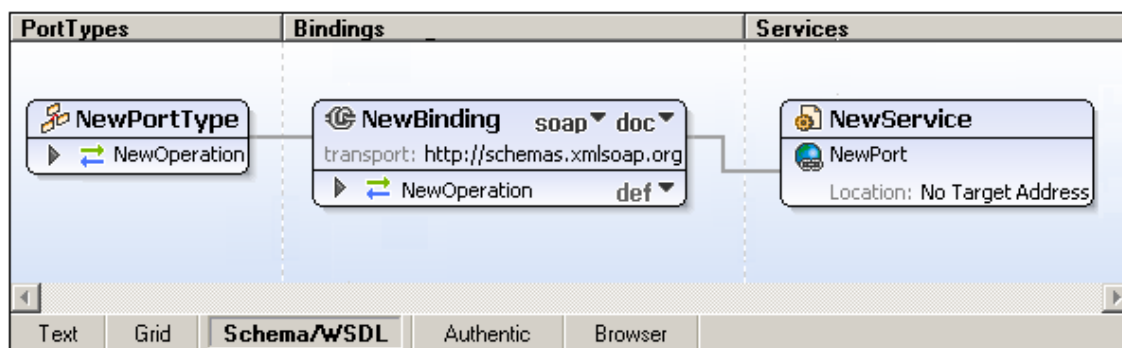
This tutorial is divided into two parts:

- In the first part, we show how a WSDL 1.1 document is created in the graphical WSDL View of XMLSpy. In this part, you will: (i) create a rudimentary WSDL 1.1 document using the **File | New** menu option; (ii) create a PortType; (iii) create a binding; (iv) create a service and a port; (v) validate the document and save it.
- In the second part, we show how to connect to a web service, save the WSDL file locally, and send a SOAP request to the web service

You can do all this in the graphical [WSDL View](#) and do not have to use the Text View. You can directly manipulate the WSDL components using drag and drop, as well as enter values of properties in the Entry Helpers of the WSDL View.

10.1.1 Creating a New Document

To create a new WSDL document, select the **File | New** command. In the Create New Document dialog that pops up, select **WSDL** (WSDL Web Service Description v1.1) as the type of document you wish to create and click **OK**. This creates a skeleton new document (*screenshot below*), which opens in the graphical WSDL View (called WSDL View in this tutorial).



Assigning a target namespace

Switch to Text View. The start tag of the `wsdl:definitions` element will look something like this:

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
3      xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
4      xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
5      xmlns:xs="http://www.w3.org/2001/XMLSchema"
6      xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
7      xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
8      xmlns:tns="http://new.webservice.namespace"
9      targetNamespace="http://new.webservice.namespace">
10

```

Change the target namespace (`targetNamespace`) attribute to `http://mywebservice.namespace` or anything else you like (since this tutorial focuses on showing how to create a WSDL document and does not provide an actual service). You should then also change the namespace value of the `tns` attribute to `http://mywebservice.namespace` (or the namespace you selected for the target namespace).

Note: In the skeleton starter document, WSDL elements are in the target namespace while

references to WSDL elements are made using the `tns` prefix. For example:

```
<wsdl:binding name="NewBinding" type="tns:NewPortType">. In order for the  
tns prefix to match the target namespace, its namespace value should be identical with  
the target namespace.
```

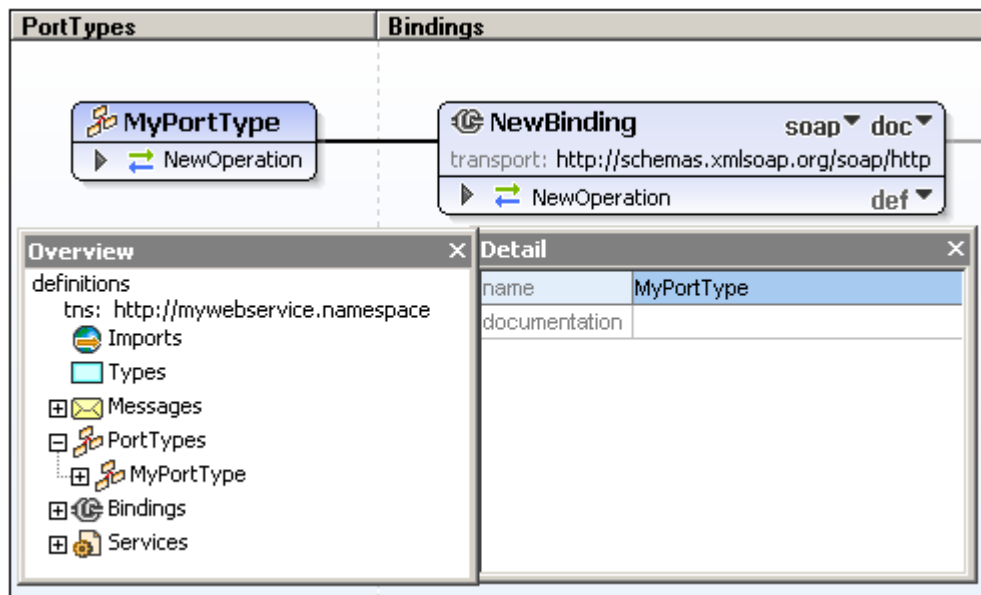
10.1.2 Creating a PortType

Creating a PortType involves the following:

- Naming the PortType
- Inserting an operation
- Adding input and output messages
- Adding parameters to messages

Naming the PortType

Rename `NewPortType` to `MyPortType` by double-clicking in the title bar of the `NewPortType` box in the design, then editing the name and pressing **Return**. Notice that the name of the PortType also changes in the Overview and Detail entry helper (screenshot below).



Inserting an operation

In the case of `MyPortType`, an operation, `NewOperation`, is already present, so we will work with this. Start by renaming `NewOperation` to, say, `EchoString` (double-click its name, edit, and press **Return**). (To insert additional operations for a PortType, right-click the PortType box, select **Append Operation**, and then click the required type of operation.)

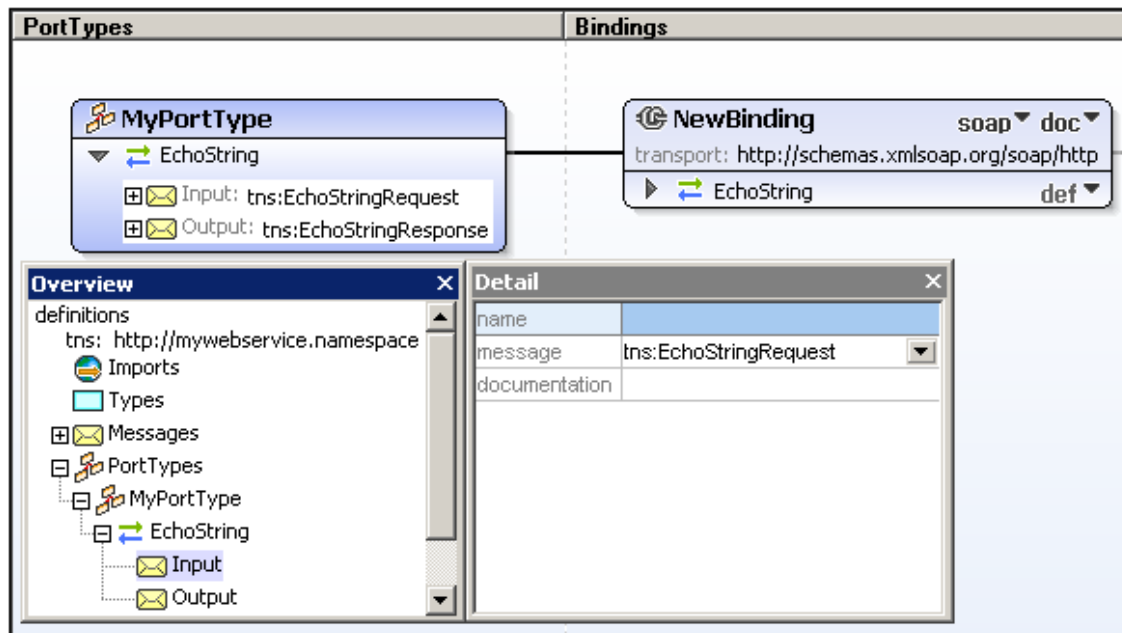
Adding input and output messages

When an operation is appended to a PortType, you can select whether the operation should be one of five types:

- Request response
- Solicit response
- One-way
- Notification

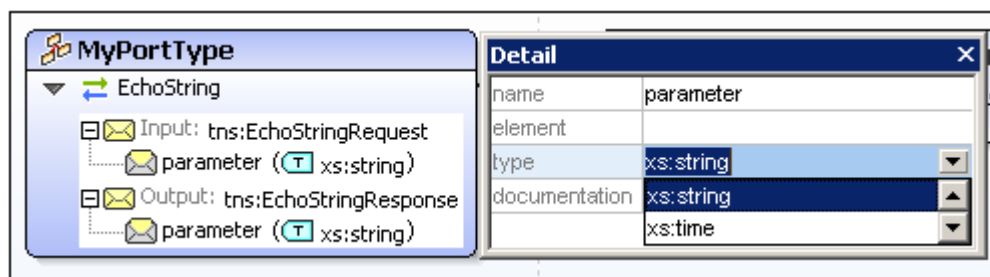
- Empty operation

For each type, input and output messages are added automatically according to the operation type. When **Empty operation** is selected, right-clicking the operation allows you to select a message type to insert. A message can be deleted by right-clicking and selecting **Delete input/output/fault element**. In the case of the `EchoString` operation, rename the input and output messages to `EchoStringRequest` and `EchoStringResponse`, respectively.



Adding parameters to messages

Each input or output message is created with a single default message part (or parameter) of type `xs:string` (see screenshot below). To add another parameter, right-click either the message or one of its parameters and select **Add message part (parameter)**.



To edit a parameter do one of the following: (i) double-click the text to edit it; or (ii) right-click the parameter and select **Edit**, or (iii) use the Detail entry helper (see screenshot above).

10.1.3 Creating a Binding

A binding is a concrete protocol and data format specification for a particular PortType. Creating a binding therefore involves the following:

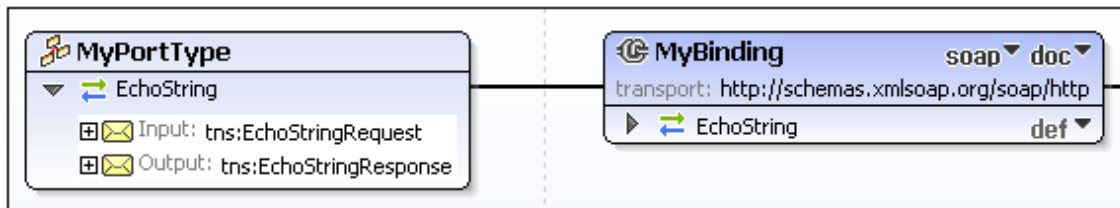
- Associating the binding with a PortType.
- Defining the binding's protocol and data format specification.
- Associating the binding with a port.

Associating a binding with a PortType

When a new binding is created (to create one, right-click anywhere in an empty area of the design and select **Insert Binding**), it has no PortType associated with it (*screenshot below*). (If you have created a new WSDL document, the binding created by default will be associated, by default, with the PortType that was created by default, and the association will be shown by a line joining the two boxes.).

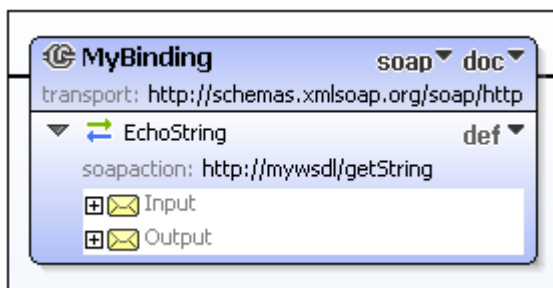


To associate a PortType with a binding, in the new Binding box click the Down arrow of the PortType entry (see *screenshot above*). This pops up a list of PortTypes defined in the document. Select the PortType with which you wish to associate the binding. When a PortType has been associated with the binding, the association is indicated by a line joining the box of the selected PortType to the Binding box, like this:



Selecting the protocol and data format

The protocol of the binding is selected by clicking the down arrow in the title bar of the Binding box (that of the `soap/http` entry), and selecting one of the four available protocols: SOAP, SOAP 1.2, HTTP-GET, and HTTP-POST (*screenshot below*). When the SOAP 1.1 or 1.2 protocol is chosen, you can select `document` or `rpc` as its data format (using the list options list popped up by the dropdown arrow to the right of the protocol selection).



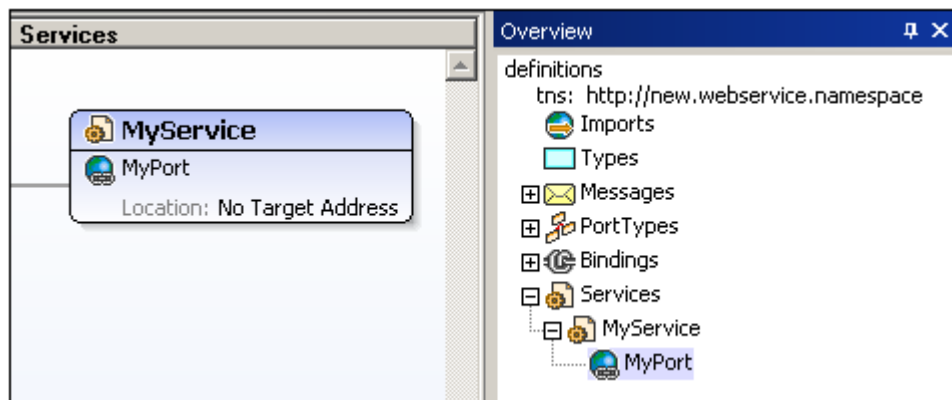
The `soapAction` for each operation in the binding can be defined in the design (see *screenshot above*) or in the Detail entry helper when that operation is selected.

Associating the binding with a port

To associate the binding with a port, the port has to be first defined. How to create a port within a service and associate a port with a binding is described in the section [Creating a Service and Ports](#).

10.1.4 Creating a Service and Ports

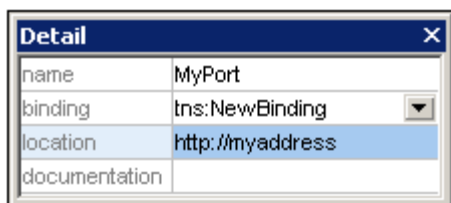
To add a new service, right-click in an empty space of the design and select **Insert Service** from the context menu. If you have created a new WSDL document, a service will already be present in the design. You can rename the service by double-clicking in its name, editing the name, and pressing **Return**. Notice that the name of the service also changes in the Overview entry helper (*screenshot below*).



In the Overview entry helper, double-click the `NewPort` entry, change it to `MyPort..`, and press **Return**. Notice that the name of the port also changes in the `MyService` box in the design (*screenshot above*). To add additional ports, right-click either the service or the port, and, from the context menu, select **Insert Port**.

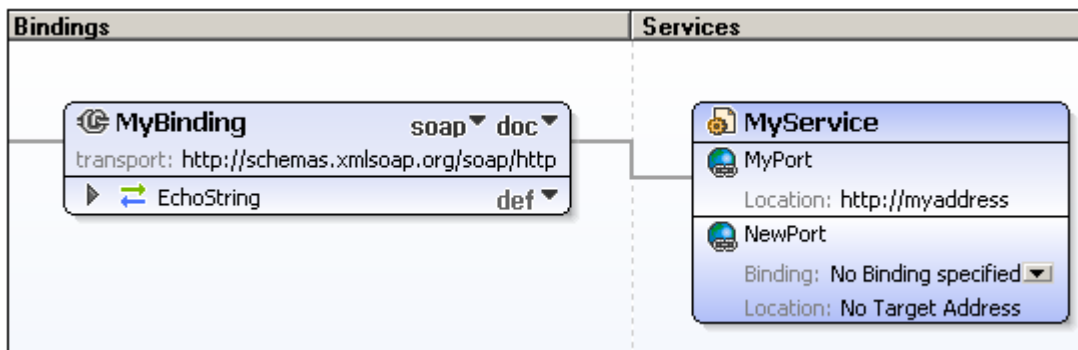
Entering the address of a port

The address of a port can be entered either: (i) directly in the design, as the value of the `Location` item (see *screenshot above*), or (ii) in the Detail entry helper (by double-clicking in the `Location` field and entering the address (*screenshot below*)).



Associating a binding with a port

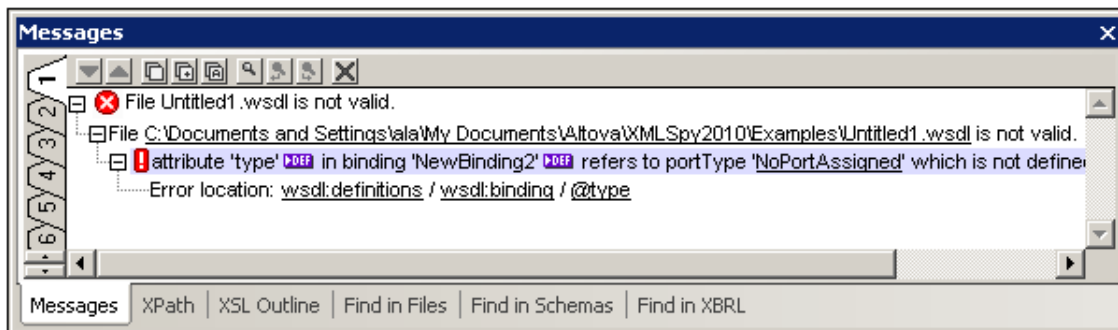
A port is the endpoint that combines a binding with a network address. Once a port's address has been defined, all that needs to be done is associate a binding with the port. To associate a binding with a port, click the down arrow of the Binding item of a port and select from the list of bindings defined in the document.



Note: If a binding is already associated with a port and you wish to associate another binding, you have to remove the binding reference (using the port's right-click menu), and then insert the new binding reference.

10.1.5 Validating the WSDL Document

After completing the WSDL document, it can be validated by selecting the **XML | Validate XML (F8)** command. The results of the validation are displayed in the Validation output bar below the Main Window (*screenshot below*).



Detailed information about any error detected is displayed, enabling you to quickly locate the error and fix it.

10.1.6 Connecting to a Web Service and Opening Files

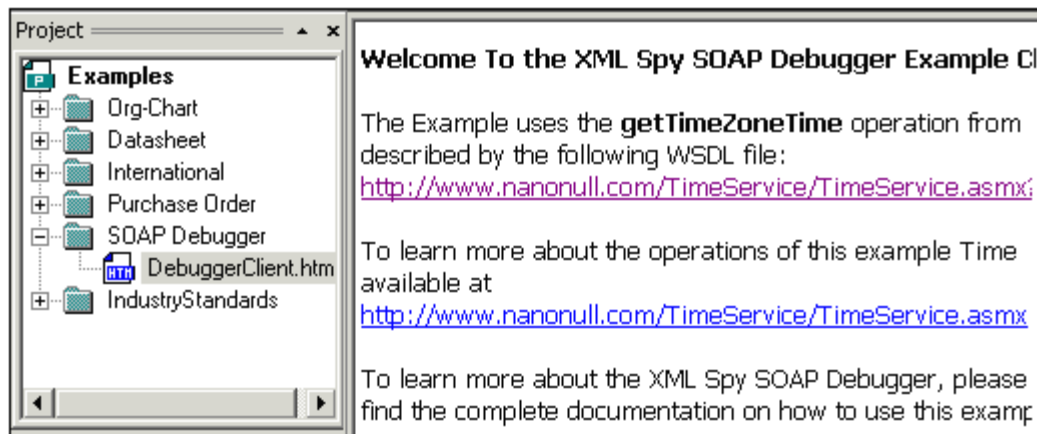
In this section, you will learn how:

- A web service can be accessed using XMLSpy
- A WSDL file on the web can be opened in XMLSpy
- An XML Schema associated with the WSDL document can be opened in XMLSpy

Accessing a web service

A web service is typically accessed via an HTML page. One such page is `DebuggerClient.htm`, which is in the project `Examples/SoapDebugger`. To access the web service displayed on this page, do the following:

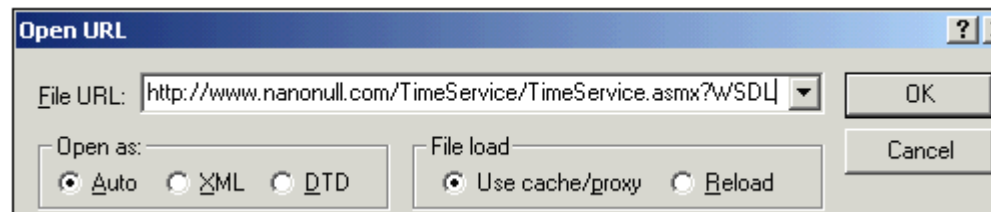
1. Activate the Project window if it is not visible (using the menu option **Window | Project window**).
2. Click the expand icon next to the SOAP Debugger folder, and double-click the file `DebuggerClient.htm`. This opens the SOAP Debugger Example Client in the Main Window.



Opening a WSDL file in XMLSpy

To open a web-based WSDL file in XMLSpy, do the following:

1. Select the menu option **File | Open** and, in the Open dialog, click the **Switch to URL** button. Then enter or copy the address `http://www.nanonull.com/TimeService/TimeService.asmx?WSDL` into the File URL field of the dialog box.



2. Click **OK** to load the WSDL file. The WSDL file is displayed in Text View.
3. Select the menu option **File | Save As**, and save the file locally, naming it, say, `timeservice.wsdl`.
4. Click the WSDL View tab to display the file in the graphical WSDL editor.

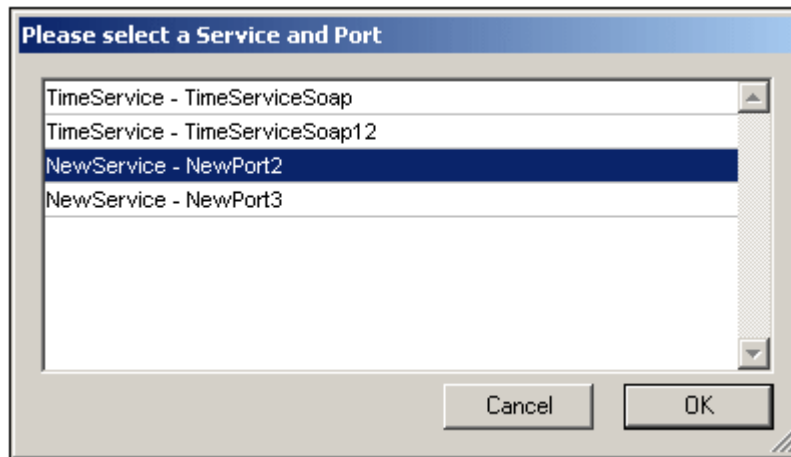
Viewing the schema file associated with the active WSDL file

With the `timeservice.wsdl` file as the active document in WSDL View, select the menu option **WSDL | Types | Edit Schema(s) in Schema View**. This opens the schema file that defines all the datatypes used in the `timeservice.wsdl` file. You can modify the schema and save changes. These changes will take effect as soon as the WSDL file is re-parsed.

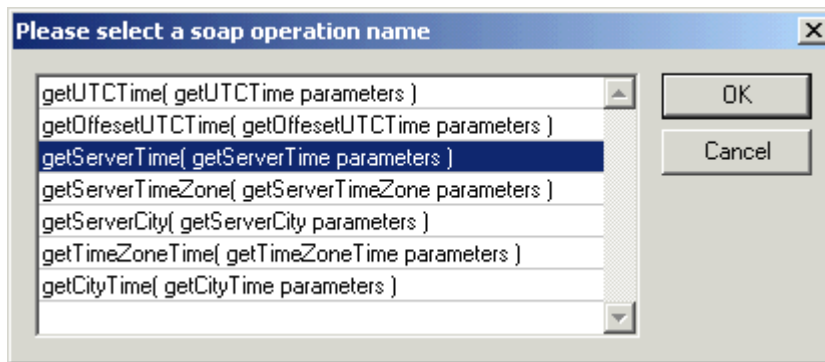
10.1.7 Sending a SOAP Request from the WSDL File

To send a SOAP request from the `timeservice.wsdl` file, do the following:

1. Make `timeservice.wsdl` the active file in the Main Window.
2. Select the menu option **SOAP | Create new SOAP request**.
3. Browse for the file `timeservice.wsdl` and confirm with **OK**.
4. If, among the various services defined in the document, there is more than one port that references a SOAP 1.1 or 1.2 binding, then a popup appears (*screenshot below*) prompting you to select the required Service and Port. After making the selection, click **OK**.



5. In the dialog box that then pops up (*screenshot below*), select a SOAP operation, for example, `getServerTime`, and click **OK**.



- This creates a SOAP request document containing the `getServerTime` operation.
6. Make the request document and select the menu option **SOAP | Send request to server**. The SOAP response document appears in the Main Window, containing the element `getServerTimeResult`, which displays the current server time of the Nanonull.com time service.

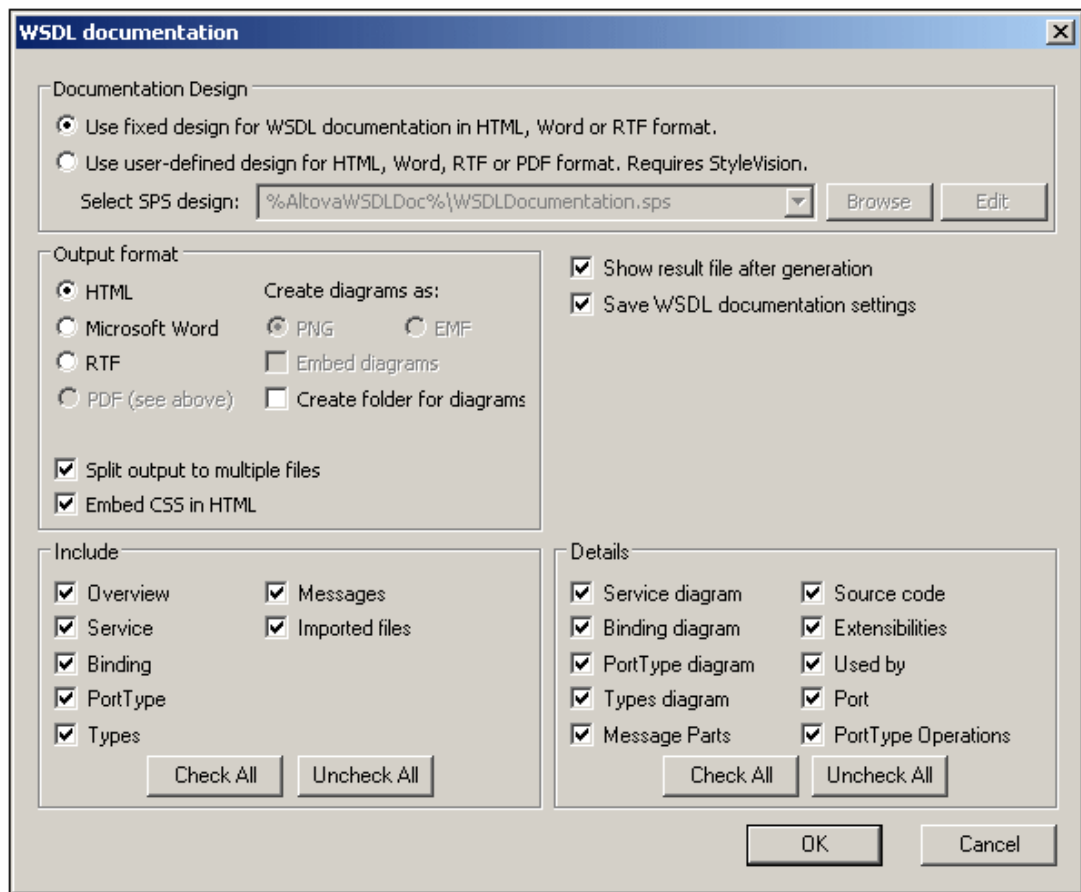
10.1.8 Creating WSDL Documentation

The **WSDL | Generate Documentation** option allows you to produce detailed documentation of the current WSDL document. You can output the documentation as an HTML, MS Word, or RTF file and specify the components you want to include. Related WSDL elements are hyperlinked in the generated documentation, allowing easy navigation.

Note: In order to generate documentation in MS Word format, you must have MS Word (version 2000 or later) installed.

To generate documentation for the WSDL file, do the following:

1. Make `timeservice.wsdl` the active document.
2. Switch to **WSDL** view.
3. Select the menu option **WSDL | Generate Documentation**.
This opens the WSDL Documentation dialog box (*screenshot below*).
4. Select the type of output you want to generate, HTML, MS Word, or RTF.
5. Select the specific WSDL components you want to include in the documentation, and set other options (see WSDL Documentation Options below).



6. Click **OK** and enter the name of the WSDL documentation file in the Save as dialog box.

WSDL documentation options

You can select from among the following documentation options:

- The required format is specified in the Output Format pane: either HTML, Microsoft Word, or RTF. The documentation can be generated either as a single file or be split into multiple files. When multiple files are generated, each file corresponds to a component. What components are included in the output is specified using the check boxes in the Include pane.
- The Embed Diagrams option is enabled for the MS Word and RTF output options. When this option is checked, diagrams are embedded in the result file, either in PNG or EMF format. Otherwise diagrams are created as PNG or EMF files, which are displayed in the result file via object links. When the output is HTML, all diagrams are created as document-external PNG files.
- In the Include pane, you select which items you want to include in the documentation. The Overview option lists all components, organized by component type, at the top of the file.
- The Details pane lists the details that may be included for each component. Select the details you wish to include in the documentation.
- The Show Result File option is enabled for all three output options. When this option is checked, the result files are displayed in Browser View (HTML output), MS Word (MS Word output), and the default application for .rtf files (RTF output).

10.1.9 Converting to WSDL 2.0

In XMLSpy you can easily convert an existing WSDL 1.1 document to the WSDL 2.0 format. Try this with the `TimeService.wsdl` example, as follows:

1. Make the file `TimeService.wsdl` the active file. (This file is in the `WSDL Editor` folder in the `Examples` project of XMLSpy.)
2. Click the command **WSDL | Convert to WSDL 2.0**.
3. In the Save As dialog that appears, enter the name with which you wish to save the WSDL 2.0 file, for example, `TimeService20.wsdl`.
4. The new file is generated, automatically validated, and displayed in WSDL View.

10.2 SOAP

Altova website:  [SOAP Debugger](#)

In this section you will learn how to:

- Validate SOAP messages against WSDL. SOAP messages can be checked for validity not only against the SOAP specification but as well as against any XML Schemas referenced in the corresponding WSDL definition
- Send and receive SOAP requests using the SOAP debugger
- Set breakpoints for sending and receiving SOAP requests
- Edit an incorrect SOAP request before sending it on to the web service

When developing an application that accesses data from a web service and does further data processing, something might happen that you don't expect. This chapter explains how to see and edit the data that is actually transmitted. This is done using the SOAP debugger. It works as a **proxy server** between your client and the web service. The debugger functionality goes a lot deeper than a normal trace utility that only stores every call's Request and Response.

SOAP debugger allows you to set breakpoints for every Request and Response message and even define Conditional breakpoints via XPath expressions.

SOAP debugger supports:

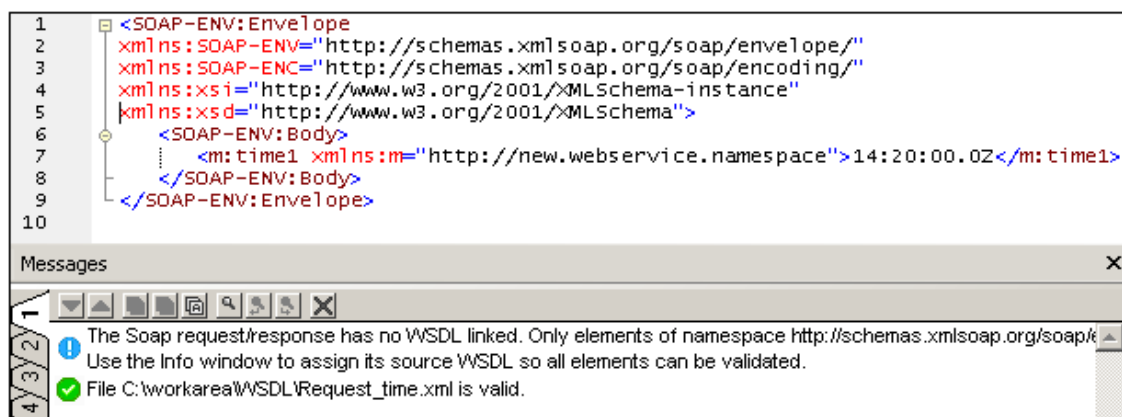
- Stepping through SOAP requests and responses
- Modification of the SOAP requests and responses
- Forwarding of modified requests to the client or server

10.2.1 SOAP Validation

SOAP messages can be checked for validity not only against the SOAP specification but also against any XML Schemas referenced in the corresponding WSDL definition.

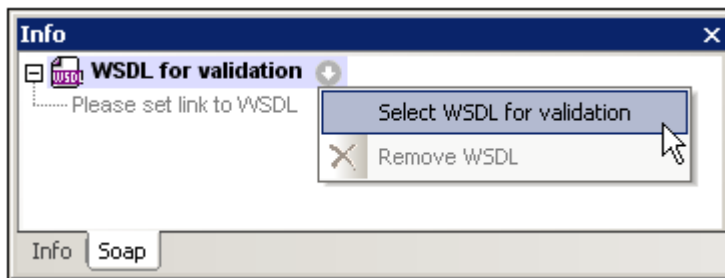
Validating against SOAP rules only

To validate a SOAP message, open the SOAP message file (*screenshot below*) and press **F8** (or the menu command **XML | Validate**). Since no WSDL file has been linked to the SOAP message file, the SOAP message is validated according to the rules for SOAP messages. The file is found to be valid if it is valid according to these rules (*see the Messages Window in the screenshot below*).

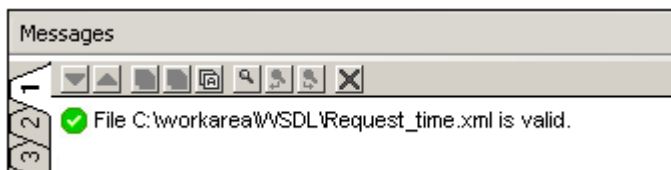


Validating against SOAP rules and linked WSDL

To validate a SOAP message additionally according to the linked WSDL, the WSDL file must be linked to the SOAP file. This is done in the SOAP tab of the Info Window (*screenshot below*). Click the button to the right of the *WSDL for Validation* item and select the command **Select WSDL for Validation**. In the dialog that pops up, browse for the WSDL file you want and click **OK**. The WSDL file will be entered in the Info Window and the SOAP message file will be linked to it.



On pressing **F8** (or the menu command **XML | Validate**) the SOAP message will be validated not only against the rules for SOAP messages but also against the rules in the linked WSDL file.



The file is found to be valid if it is valid according to both sets of rules (*see screenshot above*).

10.2.2 SOAP Communications Process

Once the proxy server (SOAP debugger) has been started, the SOAP communication process is as follows:

- Proxy server listens continually to a socket/port for incoming **client requests**
 1. Client application sends a request to proxy server
 2. Client requests can be modified if/when breakpoints have been triggered
 3. Proxy server request data is forwarded to the web service server
- The **webservice server responds** to the proxy request, and sends the response data back to the proxy server
 4. Server responses can be modified if/when breakpoints have been triggered
 5. Proxy server response data is forwarded to the client application
- Client application receives response data from proxy server

How to... example client and server applications:

- The **Client** application, which sends and receives SOAP messages is the Browser window
- The "Nanonull Time Service" service will act as the **web service server**.
The URL for this web service is:
<http://www.nanonull.com/TimeService/TimeService.asmx?WSDL>

Port settings

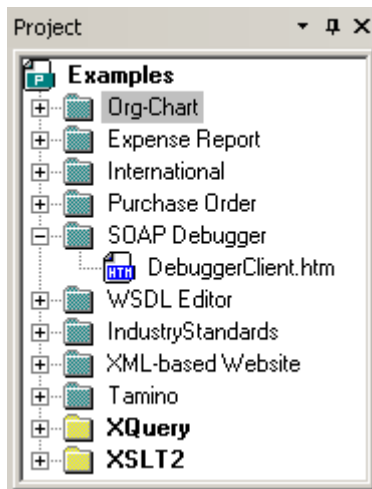
The SOAP debugger uses the 8080 port to monitor the clients' requests. The port can only be altered when a new SOAP debugging session is started.

Please note: This port may be disabled by personal firewalls, or advertising filters. You would therefore have to disable these programs, or select a different port address.

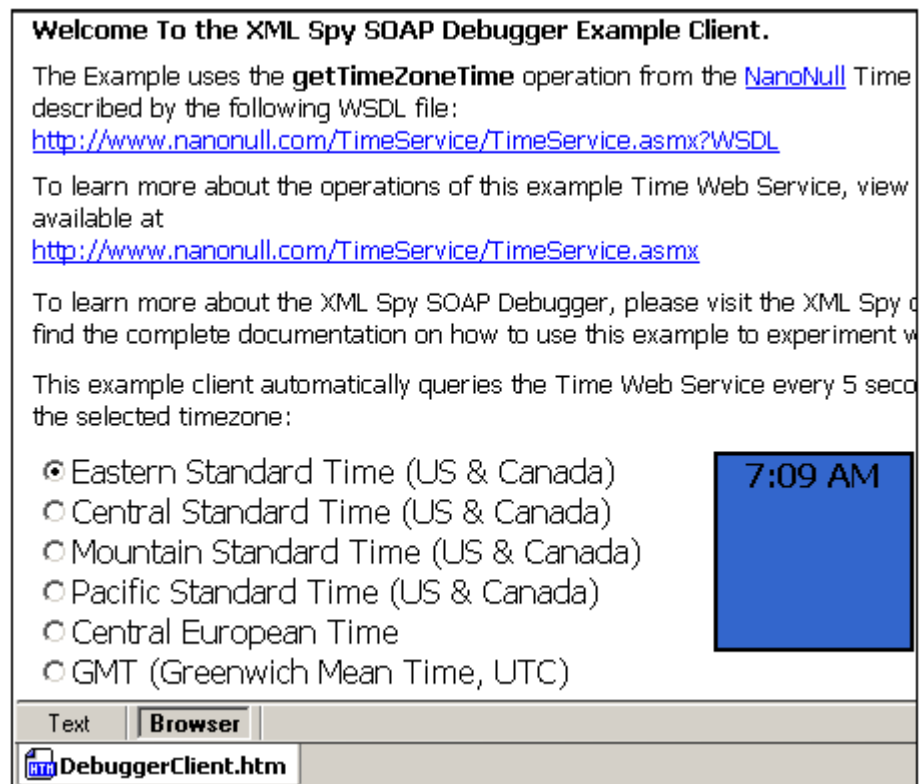
10.2.3 Using SOAP Debugger

To use the SOAP debugger:

1. Start XMLSpy.
2. Select the menu option **Project | Open Project**.
3. Select the **Examples.spp** file in the **C:\Documents and Settings\<username>\My Documents\Altova\XMLSpy2012\Examples** folder.
4. Click the + sign of the SOAP Debugger folder to see the folder contents.



5. Double-click the **DebuggerClient.htm** file to start the SOAP Client and connect to the web service.



The Browser View displays the current Eastern Standard Time (7:09 in this example).

6. Click the GMT radio button to see the current Greenwich Mean Time.

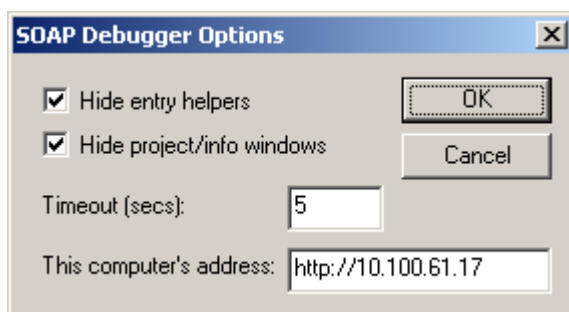


An error has obviously occurred at this point. We should have a correct time in the clock display. We will have to activate the SOAP debugger to analyze the SOAP messages, and see if we can locate the error and perhaps fix it.

Configuring the SOAP debugger

To configure the SOAP debugger:

1. Select the menu option **SOAP | SOAP debugger options**.
2. Activate the "Hide project/info windows" check box, and click **OK** to confirm.

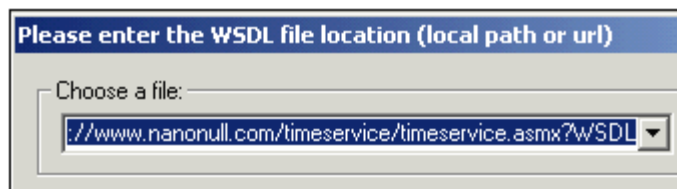


This allows more room for the SOAP request and response windows. Make sure you define these settings **before** you start the SOAP debugger session.

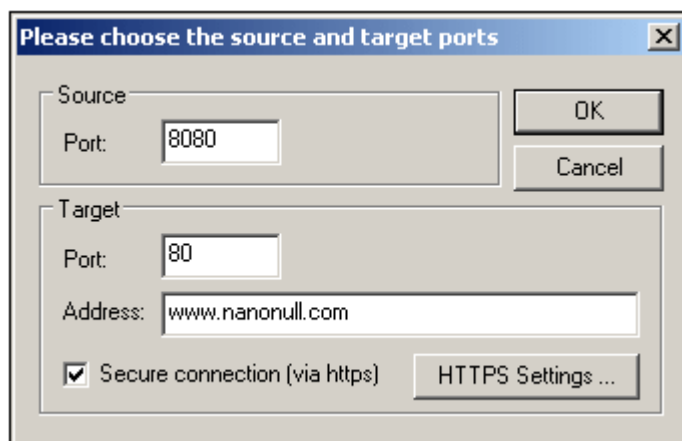
Starting the SOAP debugger

To start the SOAP debugger:

1. Select the menu option **SOAP | SOAP Debugger Session**. This opens the "Enter the WSDL file location..." dialog box.

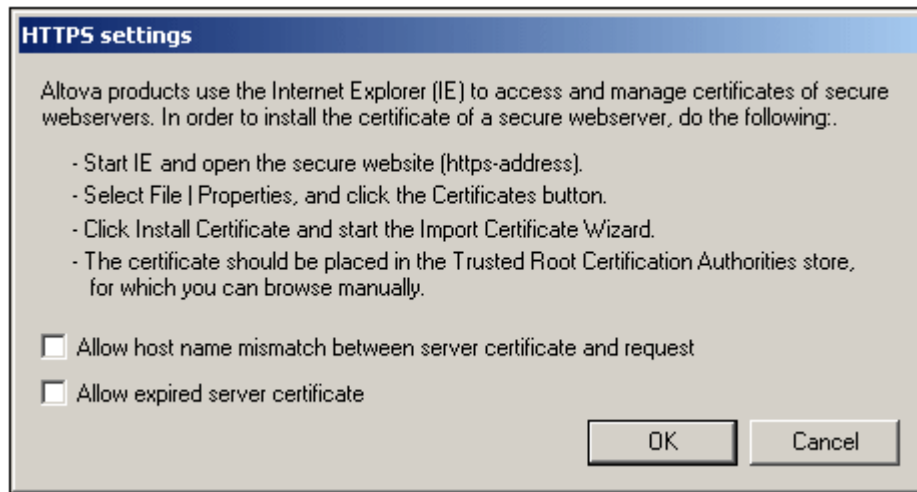


2. Enter the WSDL file location (shown below into the text box **http://www.nanonull.com/TimeService/TimeService.asmx?WSDL** and click **OK**. The following dialog box displays the source and target port settings. The **source port** setting (8080) is a default setting and is automatically entered. This setting can be changed every time the debugger is started. The **target** settings are supplied by the WSDL file, and also appear automatically. If the port number of the target ends with 443, the *Secure Connection (via https)* check box will be checked by default. If any other port number is entered, then you must check the *Secure Connection* check box manually. Click the **HTTPS Settings** button to select the secure connection options (see the section [Change SOAP Request Parameters](#) for details about using these options).



3. Click **OK** to start the SOAP debugger session. This opens the SOAP debugger proxy server in its **inactive** state. You can see this by noting that the proxy server icon  is

disabled. If you are using the `https` protocol, you might wish to allow a host name mismatch or an expired server certificate. To make these settings, click the **HTTPS Settings** button. The HTTPS Settings dialog (*screenshot below*) appears.



4. In the HTTPS Settings dialog, you can allow that the host name given in the SOAP request need not match the host name in the server certificate; do this by checking the *Allow host name mismatch* check box. If you wish to allow the connection despite an expired server certificate, then check the *Allow expired server certificate* check box. Confirm the changes with **OK**.

About trusted certificates

Altova products use Internet Explorer (IE) to access and manage trusted certificates of secure webserver. Installing the certificate of a webserver in IE allows IE to access the webserver without issuing a warning or aborting the process. In order to install the certificate of a secure webserver, do the following:

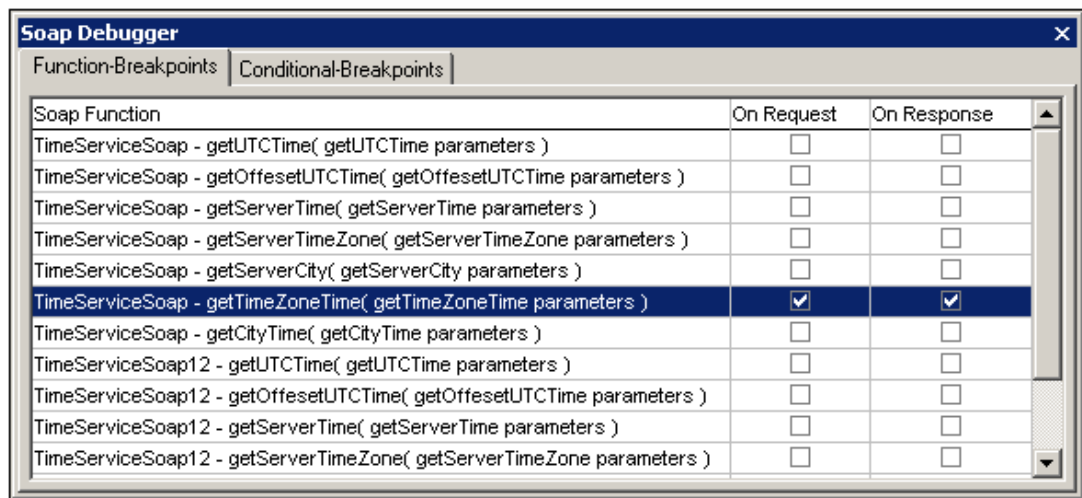
- In Internet Explorer 8, open the secure website.
- Select **File | Properties**, and click the Certificates button.
- Click **Install Certificate** and start the Import Certificate Wizard. (This Wizard can also be accessed via **Tools | Internet Options | Content | Certificates | Import.**)
- The certificate should be placed in the Trusted Root Certification Authorities store, for which you can browse manually.
- Finish the Wizard steps, close the Certificates and Properties dialogs respectively by clicking **OK**. You might need to restart Internet Explorer.

Setting debugger breakpoints

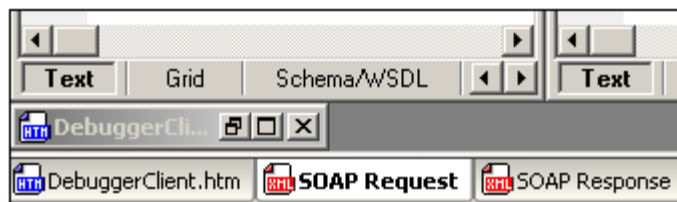
To set debugger breakpoints:

This web service uses the method **getTimeZoneTime** to find the time zone time.

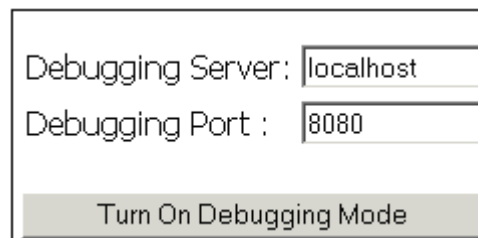
1. Activate the **On Request** and **On Response** check boxes for this method (**getTimeZoneTime**). This enables you to analyze both SOAP requests and responses for errors.



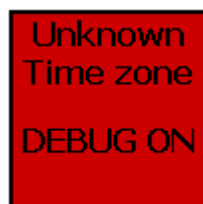
- Click the **GO** icon  (or use the menu command **SOAP | GO**).
- Click the **DebuggerClient** tab to switch to the Client. You may have to select **Window | DebuggerClient.htm** to see the contents.



- Click the **Turn On Debugging Mode** button in the client window.



This displays a "Debugging is on" message, and sends the SOAP request to the debugger.



Viewing and editing the SOAP request

The SOAP request automatically appears in the SOAP request window of the debugger, in Text view. We can now look at the request and edit any errors it might contain.

```
<?xml version='1.0' encoding='utf-8'?>
<SOAP-ENV:Envelope xmlns="" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
<SOAP-ENV:Body>
<getTimeZoneTime xmlns="http://www.Nanonull.com/TimeService/"
<timezone>GMD</timezone></getTimeZoneTime></SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Looking at the **timezone** element, we notice that the value is **GMD**. This cannot be correct, so we will change it to GMT and see what happens.

1. Double-click in the **timezone** field, and change the entry to **GMT**.

```
<?xml version='1.0' encoding='utf-8'?>
<SOAP-ENV:Envelope xmlns="" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
<SOAP-ENV:Body>
<getTimeZoneTime xmlns="http://www.Nanonull.com/TimeService/"
<timezone>GMT</timezone></getTimeZoneTime></SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

2. Click the **GO** icon  (or use the menu command **SOAP | GO**) to send the **corrected request** to the web service.

After a few seconds, the **web service response** to the SOAP request appears in the SOAP response window. Select **View | Word wrap** to see the contents as displayed in the diagram below.

```
1 <?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:soap="
http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsi="
http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="
http://www.w3.org/2001/XMLSchema"><soap:Body><
getTimeZoneTimeResponse xmlns="http://www.Nanonull.com/TimeService/"><
getTimeZoneTimeResult>3:06 PM</getTimeZoneTimeResult></
getTimeZoneTimeResponse></soap:Body></soap:Envelope>
```

Text Grid Schema/WSDL Authentic Browser

DebuggerClient.htm SOAP Request SOAP Response

We did not receive an error message in the response, so we can assume that this is the correct time.

3. Click the debugger **GO** icon  to send this response to the **client** (the Browser window in this case).
4. Click the **DebuggerClient.htm** tab to switch back to the client, and see the new result.

☐ Eastern Standard Time (US & Canada)
☐ Central Standard Time (US & Canada)
☐ Mountain Standard Time (US & Canada)
☐ Pacific Standard Time (US & Canada)
☐ Central European Time
☒ GMT (Greenwich Mean Time, UTC)

3:06 PM
DEBUG ON

The error message has disappeared, and the correct GMT time is displayed.

5. Select the menu option **SOAP | SOAP debugger session** to close the debugger

session.

Fixing the error

Now that we know that the **client** is actually sending the invalid value "GMD" instead of "GMT" in its request and the server would correctly respond to "GMT", we have successfully identified the SOAP Client as being the component that causes the error, so we can go about fixing it.

In the **DebuggerClient.htm** window:

1. Click the Text View button, open the Find dialog (**Edit | Find**), and search for "GMD". XMLSpy will display this code fragment:

```

138      else if(timezone[4].checked)
139      {
140          msCurrentTimeZone='CET';
141      }
142      else if(timezone[5].checked)
143      {
144          msCurrentTimeZone='GMD';
145      }

```

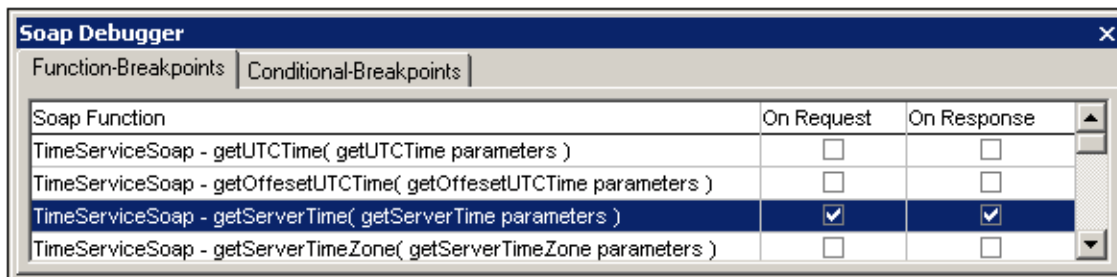
2. Change the value from "GMD" to "GMT" and switch back to the Browser View. The SOAP Client now works correctly, the "Unknown Time zone" message has gone and the correct time appears in the window.

Closing SOAP Debugger

To close the SOAP Debugger session, click the command **SOAP | SOAP Debugger Session**.

10.2.4 Breakpoints in SOAP Debugger

The SOAP Debugger window is where you set and delete breakpoints. It is separated into two tabs (*screenshot below*).



Function-Breakpoints tab

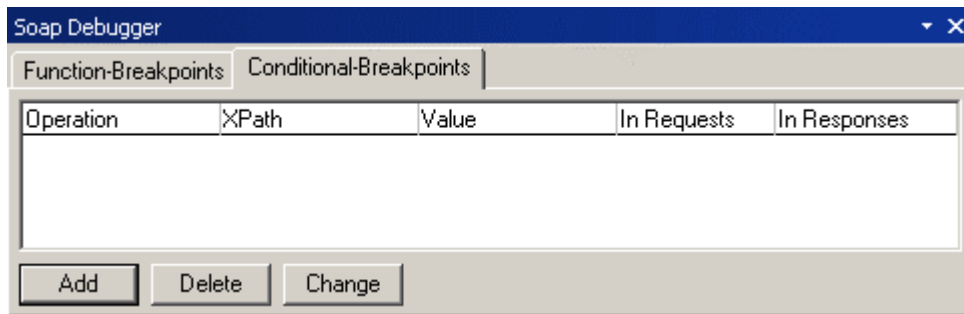
The Function-Breakpoints tab allows you to set a breakpoint on Requests and/or Responses to SOAP methods. The debugger highlights the line of the function which triggered the breakpoint. Data packets to and from the client are analyzed and matched to the corresponding functions from the WSDL file. If a breakpoint is set for a specific method, then this is where the SOAP debugger stops. The toolbar buttons are enabled at this point.

The data is displayed in the Soap Request or Soap Response document window. The SOAP documents displayed in the SOAP windows can be modified at this point. The data is sent the moment you click one of the toolbar icons (except for the Stop Server icon).

Conditional-Breakpoints

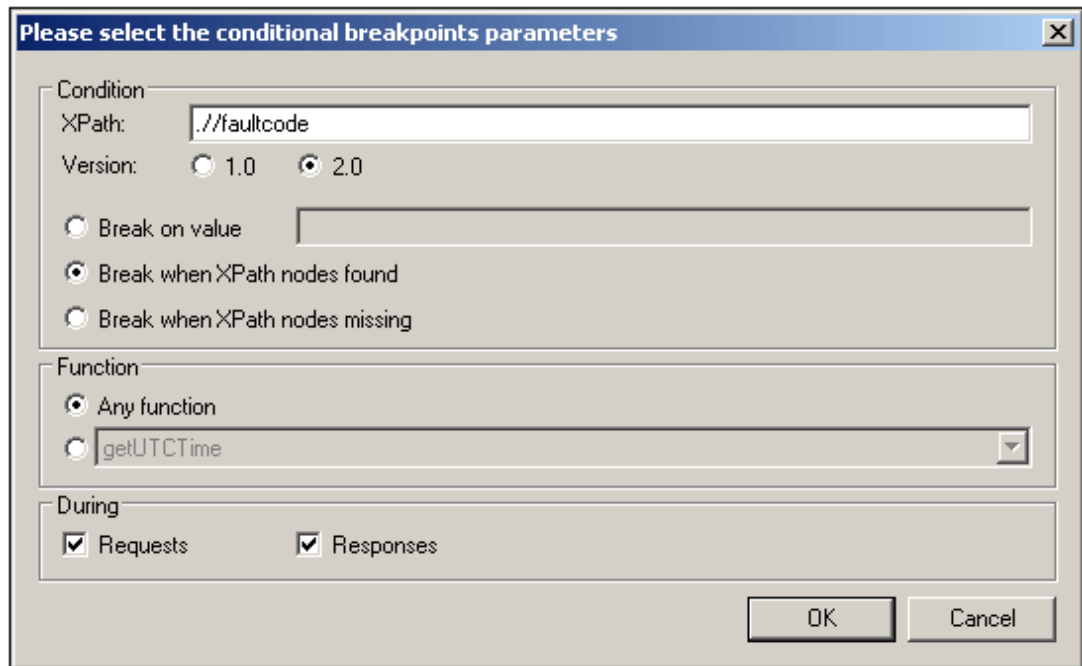
The Conditional-Breakpoints tab (*screenshot below*) allows you to use XPath expressions to define breakpoints. If a SOAP request causes an error, the SOAP response must contain a

`faultcode` element. We therefore would like to have a breakpoint triggered whenever a `faultcode` element appears.



To add a conditional breakpoint, do the following:

1. Click the Conditional Breakpoints tab, and then the **Add** button. The dialog shown below appears.



2. Enter the XPath expression (for example, `. //faultcode`) in the XPath field.
3. Select the required XPath version (1.0 or 2.0) and the *Break when XPath nodes found* radio button.
4. Click **OK** to confirm the settings. The SOAP debugger will stop whenever a `. //faultcode` element appears in a SOAP request or response.

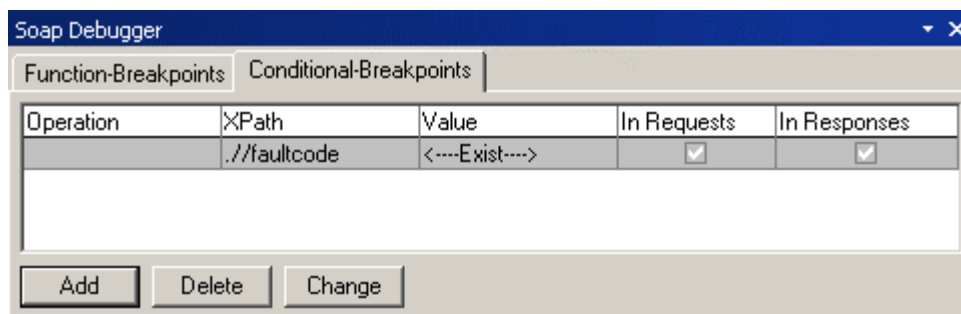
The various options in this dialog are described below:

- *XPath expression field:* Enter the specific XPath expression/node here. An XPath has to be entered here to be able to use any of the specific radio button options.
- *Version:* The XPath version you wish to use for the XPath expression.
- *Break instruction radio buttons:* The debuggers stops when the selected option occurs. The available options are: (i) Break when the targeted XPath node matches the value entered in this field; (ii) Break when the specified XPath node exists in the SOAP request or response; and (iii) Break when the specified XPath node does not exist in the

SOAP request or response.

- *Requests and Responses*: Specifies whether the options in the dialog are to be applied in SOAP responses and/or requests.
- *Functions*: Either all methods/functions are scanned for the condition you define (*Any function* radio button) or you enter a specific method/function to scan.

For the condition defined in the dialog displayed above, the following conditional breakpoint will be listed in the Conditional-Breakpoints tab.



Given below is a description of the columns in this tab:

- The *Operation* column contains the method/function being searched. If you selected the *Any function* radio button then this field remains empty. If you selected a specific method/function, then this method/function is displayed here.
- The *XPath* column contains the XPath expression you defined.
- The *Value* column contains the XPath value against which the nodes returned by the XPath expression are checked for a match. If you selected *Break on value*, the specific string you entered is displayed here. If you selected *Break when XPath nodes found*, then <---Exist---> is displayed. If you selected *Break when XPath nodes missing*, then <---Missing---> is displayed.
- The *In Requests* and *In Responses* check boxes indicate where the condition is checked. You can change the settings by directly clicking the check box in the column.

To edit a conditional breakpoint, double-click its line in the tab or click the **Change** button (see *screenshot above*). To delete a conditional breakpoint, select the line you want to delete and click **Delete**.

11 XBRL

Altova website:  [XBRL Taxonomy Editor](#), [XBRL Validator](#)

XMLSpy's [XBRL View](#) is an XBRL taxonomy editor that provides a graphical overview of XBRL taxonomies and intelligent taxonomy editing features. In this section we describe how to create and edit taxonomies in XBRL View.

This section is organized as follows:

- It starts with a brief look at the distinction between [new and existing taxonomies](#) and the significance of this distinction. This is followed by an explanation of [the files that constitute an XBRL taxonomy](#) and how these are displayed in XBRL View.
- Starting with the section [Creating a New Taxonomy](#), we describe, step-by-step, how to build a taxonomy in XBRL View. How to use the New Taxonomy Wizard is also explained in the process. At the end of each section is a set of instructions to guide you with the creation of your own taxonomy in XBRL View. Each set of instructions helps you put into practice or test the information given in that section, and it builds upon the taxonomy created till that point using instructions in previous sections.

For more related information, see the sections: [Editing Views | XBRL View](#) and the description of commands in the [XBRL menu](#). For example, information about generating documentation for the taxonomy (as seen in XBRL View) will be found in the section User Reference | XBRL Menu | Generate Documentation.

[XML signatures](#) for XBRL files in XBRL View can be created as external signature files. How to work with signatures is described in the section, [XML Signatures](#).

Note: XMLSpy supports US-GAAP, including US-GAAP 2011.

11.1 Taxonomies: New and Existing

In XMLSpy's XBRL View you can edit existing taxonomies and create new taxonomies.

- *Existing taxonomies:* There are two types of existing taxonomies: (i) standard taxonomies that should not be edited; and (ii) non-standard taxonomies which may be edited; these might have been created by you or another party.
- *New taxonomies:* New taxonomies can be created in XMLSpy. These are of two types: (i) taxonomies that are created from scratch; and (ii) taxonomies that extend a standard taxonomy.

Both kinds of taxonomies can be viewed and edited in XBRL View. In some cases, such as when a standard taxonomy is imported into a taxonomy you are creating (in order to extend the imported taxonomy), you will not be allowed to edit the imported taxonomy. Elements from imported taxonomies that are not allowed to be edited are displayed in gray.

Steps for creating a new taxonomy

A new taxonomy typically will build on an existing one. In the new taxonomy, new elements will be added, and relationships between these new elements and between new elements and imported elements will be created. The general requirements of a new taxonomy and how you would go about creating one are outlined below:

1. The new taxonomy must be created in its own namespace in order to distinguish it from other taxonomies. If the new taxonomy is to extend an existing one, the existing taxonomy must be imported into the new taxonomy.
2. New concepts (elements) are defined in the new taxonomy.
3. Relationship files (or linkbases) are created to contain the definition, presentation, calculation, label, and reference relationships of the new taxonomy.
4. Relationships for the new taxonomy must be built from scratch.

In the description above, we have used the term *taxonomy* to denote the entire taxonomy, which comprises several files: the concept definitions files and the relationship files. (See the section [Taxonomy Document Files](#) for a description of the various files that comprise a taxonomy.)

Using XBRL View

In the sections that follow, we describe how to use the features of XBRL View to create and edit taxonomies. Starting with the section, [Creating a New Taxonomy](#), we provide instructions, at the end of each section, for creating your own taxonomy. The instructions in each successive section build upon the work of previous sections. By the time you reach the section [Creating Relationships: Part 1](#), you will have become familiar with XBRL View and be able to use it confidently.

The taxonomy you will be creating leads, with additional work, to the taxonomy supplied with XMLSpy (Nanonull.xsd) and which is located in the folder C:\Documents and Settings \<username>\My Documents\Altova\XMLSpy2012\Examples\XBRLExamples\Nanonull. (Note that the main taxonomy file always has the extension .xsd. The file extension .xbrl is used for XBRL instance files and not for taxonomy files.)

11.2 Taxonomy Files Overview

A well-designed XBRL taxonomy stores taxonomy concepts in a separate file from the taxonomy relationships. We call this file the main taxonomy file or the concept definitions file. Furthermore, since there are different kinds of relationships, relationships will be stored in separate files for each kind. The table below lists the different kinds of files that normally constitute a taxonomy document.

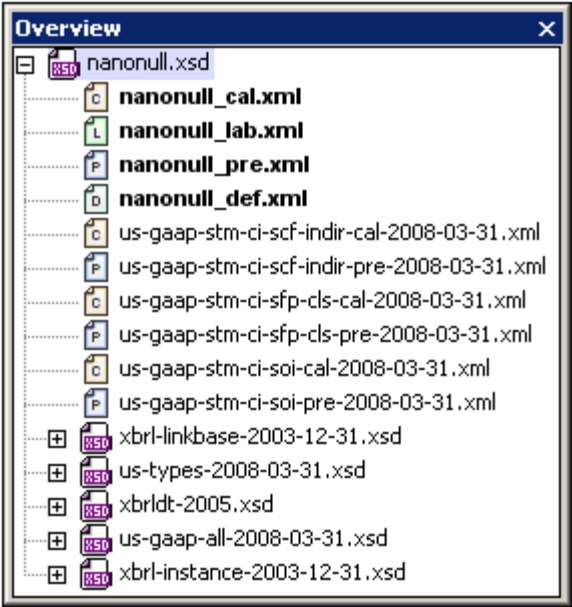
XBRL File	Description	File Type
Concepts	Each concept is defined in an XML Schema element element.	XML Schema file (.xsd)) Concept definitions file
Definition Relationships	A <code>definitionLink</code> element contains all locators and definition arcs for concept relationships.	XML file (.xml)
Calculation Relationships	A <code>calculationLink</code> element contains all the locators and calculation arcs.	XML file (.xml)
Presentation Relationships	A <code>presentationLink</code> element contains all the locators and presentation arcs.	XML file (.xml)
Labels	A <code>labelLink</code> element contains all the locators, label arcs, and labels.	XML file (.xml)
References	A <code>referenceLink</code> element contains all the locators, reference arcs, and reference resources.	XML file (.xml)

The locations of the relationship files are specified in the concept definitions file (the .xsd file) inside a `/schema/annotation/appinfo` element:

```
<xsd:annotation>
  <xsd:appinfo>
    <link:linkbaseRef xlink:arcrole="
http://www.w3.org/1999/xlink/properties/linkbase"
      xlink:href="NanonullLabels.xml" xlink:type="simple"
      xlink:role="http://www.xbrl.org/2003/role/labelLinkbaseRef" />
    <link:linkbaseRef xlink:arcrole="
http://www.w3.org/1999/xlink/properties/linkbase"
      xlink:href="NanonullDefinitions.xml" xlink:type="simple"
      xlink:role="http://www.xbrl.org/2003/role/definitionLinkbaseRef" />
    <link:linkbaseRef xlink:arcrole="
http://www.w3.org/1999/xlink/properties/linkbase"
      xlink:href="NanonullPresentations.xml" xlink:type="simple"
      xlink:role="http://www.xbrl.org/2003/role/presentationLinkbaseRef" />
    <link:linkbaseRef xlink:arcrole="
http://www.w3.org/1999/xlink/properties/linkbase"
      xlink:href="NanonullCalculations.xml" xlink:type="simple"
      xlink:role="http://www.xbrl.org/2003/role/calculationLinkbaseRef" />
    <link:linkbaseRef xlink:arcrole="
http://www.w3.org/1999/xlink/properties/linkbase"
      xlink:href="NanonullReferences.xml" xlink:type="simple"
      xlink:role="http://www.xbrl.org/2003/role/referenceLinkbaseRef" />
  </xsd:appinfo>
```

```
</xsd:annotation>
```

When the concept definitions file (the .xsd file) is open in XBRL View, the various taxonomy files are displayed in a tree structure in the [Overview entry helper](#) (screenshot below).



In the screenshot above, notice the icons to the left of the file names. XML Schema (.xsd) files are indicated with an XSD icon. The relationship files have a colored file icon with a character corresponding to the initial character of the relationship kind. For example, a icon indicates a Definition relationships file, a icon indicates a Presentation relationships file, and so on. Double-clicking any of these files opens it in XMLSpy, where it can be edited in Grid View (screenshot below) or Text View.

labelLink	
xlink:type	extended
xlink:role	http://www.xbrl.org/2003/role/link
loc	
xlink:type	locator
xlink:href	Company.xsd#company_AllProducts
xlink:label	company_AllProducts
labelArc	
xlink:type	arc
xlink:arcrole	http://www.xbrl.org/2003/arcrole/concept-label
xlink:from	company_AllProducts
xlink:to	company_AllProducts_lbl
label	
xlink:type	resource
xlink:role	http://www.xbrl.org/2003/role/label
xlink:label	company_AllProducts_lbl
xm:lang	en
Text	All Products

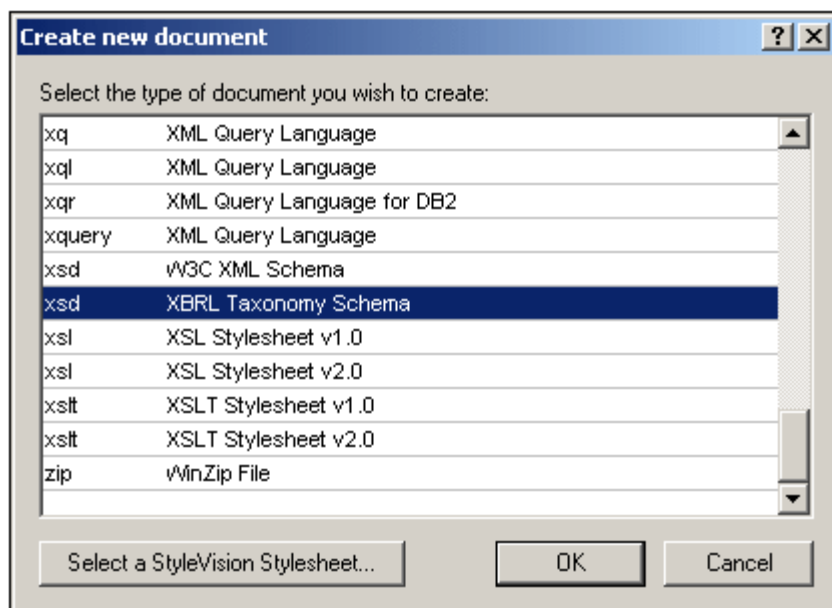
11.3 Creating a New Taxonomy

A new taxonomy would typically be created to extend one or more standard taxonomies. If a new taxonomy builds upon a standard taxonomy or an already existing taxonomy, it must import the existing taxonomy. Alternatively, a new taxonomy can be built from scratch. In XMLSpy's XBRL View, you can easily import US-GAAP and IFRS taxonomies using the New Taxonomy Wizard. The taxonomy can then be modified using the graphical interface of XBRL View.

The first step in creating a new taxonomy is to create its concept definitions file, which is an XML Schema (.xsd) file. Besides containing concept definitions, this file defines and declares the namespace of the new taxonomy, locates taxonomies to be imported, locates the relationships files of the taxonomy, and declares the namespace of imported taxonomies and other namespaces used.

Creating the concept definitions file

To create a new XBRL taxonomy, select the menu command **File | New**. This pops up the Create a New Document dialog (*screenshot below*).



Select *xsd: XBRL Taxonomy Schema* and then click **OK**. This pops up the first screen of the New Taxonomy Wizard (*screenshot below*).

Create New Taxonomy - Step 1

Ticker (or company) name:

Taxonomy base:

Date:

Destination Folder: ...

Namespace:

Schema file name:

Definition linkbase file name:

Presentation linkbase file name:

Calculation linkbase file name:

Reference linkbase file name:

Label linkbase file name:

The New Taxonomy Wizard helps you create XBRL taxonomies that are conformant with US-GAAP and IFRS requirements. Essentially it imports the taxonomies you specify. Subsequently, the file can be edited as required.

In the first dialog of the wizard, enter the following:

- The ticker symbol (or name) of the company for which the taxonomy is being created. Listed companies typically have a unique ticker symbol, and a number of important taxonomy properties and components will be named on the basis of the ticker symbol you enter. For example, the ticker symbol you enter in the *Ticker Name* field will be used to automatically generate the taxonomy namespace and the names of the taxonomy's relationship files (see *screenshot above*). If the company has no ticker symbol, you can enter in this field the name of the company for which you are creating this taxonomy.
- The taxonomy base. Options are: *US-GAAP 2011*, *US-GAAP 2009*, *US-GAAP 1.0*, *IFRS*, and *None*. The taxonomy that you select in this combo box will form the base of your taxonomy. The next dialog that the wizard displays will depend on what taxonomy base you select. The various taxonomy bases are explained in the description of the second screen of the wizard.
- The date. This can either be entered directly or be selected via a date-picker available in the dropdown of the combo box. The date, like the ticker symbol, will be used to generate the taxonomy namespace and the names of the taxonomy's relationship files.
- The destination folder is the location where the main taxonomy file and associated files will be saved.

- The taxonomy namespace (target namespace of the main taxonomy concepts file, which is a .xsd file) and the names of the concept definitions file (*Schema File*) and the of relationship files are generated automatically from: (i) the ticker symbol (or company name) and (ii) the date. If you try to edit any of these, you will receive a warning to the effect that changing the values in these fields would lead to non-conformance with the rules of the relevant taxonomy.

After you have finished, click **Next**. The dialog that is displayed next depends on what you selected as the base for your taxonomy. There are three possibilities:

- If you selected a US-GAAP taxonomy, then the US-GAAP-specific second screen of the New Taxonomy Wizard appears (*screenshot below*). You can now select the entry points you wish to include in the taxonomy, as well as specify whether the US-GAAP Core Schema should be imported (check box at the bottom of the dialog).

Create New Taxonomy - Step 2

Select the US-GAAP 2011 entry points:

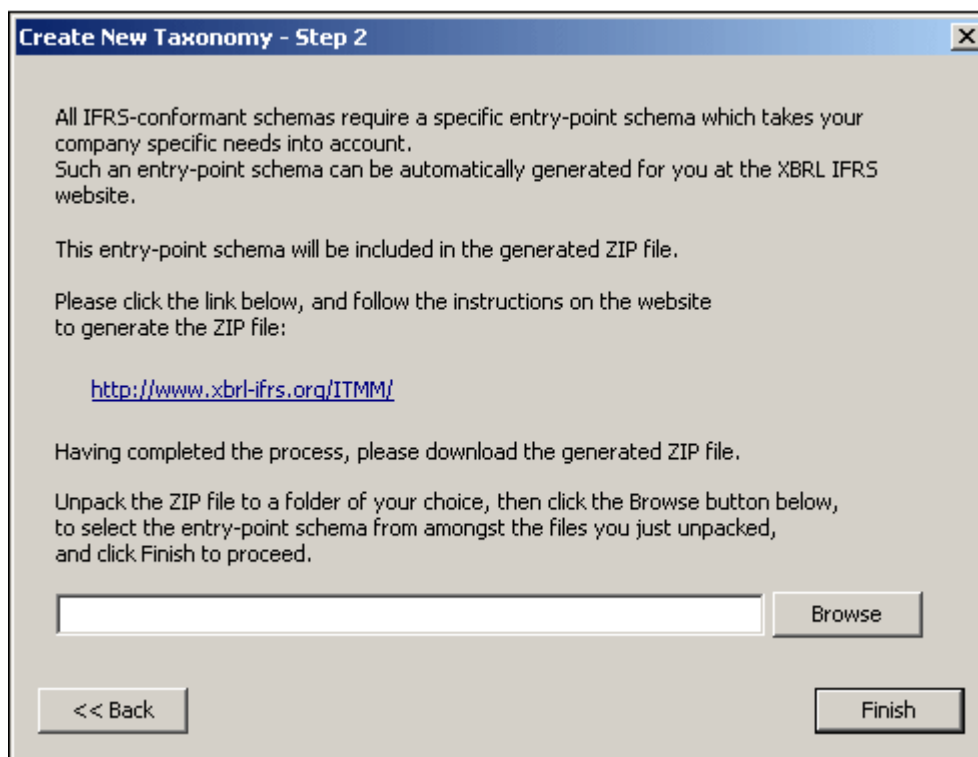
- ☐ **All Taxonomies**
- ☐ **US-GAAP 2011 Industry Entry Point Taxonomies**
 - ☐ Banking and Savings Institutions
 - ☐ Brokers and Dealers
 - ☐ Commercial and Industrial
 - ☐ Insurance
 - ☐ Real Estate
- ☐ **US-GAAP 2011 View Entry Point Taxonomies**
 - ☐ Schedule of Investments
- ☐ **Non GAAP 2011 Entry Point Taxonomies**
 - ☐ Country Code
 - ☐ Currency
 - ☐ Document and Entity Information
 - ☐ Exchange
 - ☐ Investment
 - ☐ North American Industrial Classification System (NAICS)
 - ☐ Standardized Industrial Classification (SIC)
 - ☐ State or Province

☒ Import US-GAAP 2011 Core Schema

<< Back Finish

When you click **Finish**, a new concept definitions (.xsd) file that imports the selected entry-point schemas is created in the destination folder you specified in the wizard. The taxonomy opens in XBRL View and is ready to be edited.

- If you selected IFRS as the base for your taxonomy, the IFRS-specific second screen of the New Taxonomy Wizard appears (*screenshot below*). Click the link in the dialog to go to the IFRS website, where you can generate a ZIP file containing your customized entry-point schema. Download the ZIP file and unpack it to a local folder. Browse for the entry-point schema in the local folder, select it, and then click **Finish** in the wizard.



When you click **Finish**, a new concept definitions (.xsd) file that imports the selected entry-point schemas is created in the destination folder you specified in the wizard. The taxonomy opens in XBRL View and is ready to be edited.

- If you selected **None** (that is, no base taxonomy), a new concept definitions (.xsd) file is created in the destination folder you specified in the wizard and opens in XBRL View. Notice that two elements (`xbrldt:hypercube` and `xbrldt:dimension`) are automatically created and that the basic schemas required for every taxonomy have been imported automatically (see [Overview entry helper](#)).

After you have created your base taxonomy, the next steps are: (i) to [create a namespace for the taxonomy](#) (the target namespace of the schema); and (ii) to [import the taxonomy or taxonomies](#) on which the new taxonomy is to be built. If you have used the New Taxonomy Wizard, a default taxonomy namespace will have been created and the relevant taxonomies will have been imported.

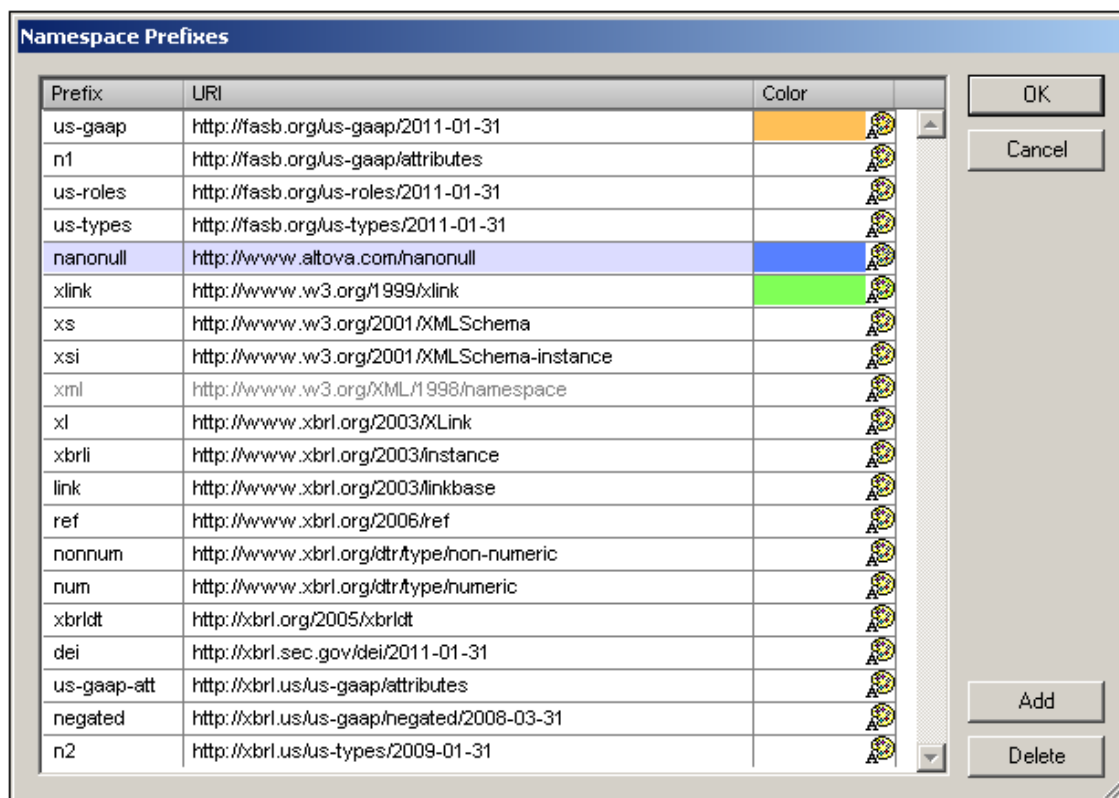
Example file: Step 1

Create a new taxonomy document using the New Taxonomy Wizard as described above (using US-GAAP 2011 as the base taxonomy and importing the US-GAAP 2011 Core Schema), and save the taxonomy with any name to a suitable location. This file is the main taxonomy file, or concept definitions file. It is an XML Schema file and must have a .xsd file extension. We will refer to the file we are creating as `Nanonull.xsd`. This is the same name as that of the supplied example in `C:\Documents and Settings\<username>\My Documents\Altova\XMLSpy2012\Examples\XBRLExamples\Nanonull`. We have also simplified the names of the relationship files by leaving out the date-generated part of their names.

In the next step we will [modify the target namespace](#) of the taxonomy that was automatically created by the New Taxonomy Wizard.

11.4 Namespaces in the Taxonomy

Taxonomy namespaces can be managed in the Namespaces dialog (*screenshot below*), which is accessed in XBRL View via the menu command **XBRL | Namespace Prefixes**. In the dialog, you can declare namespaces and associate prefixes and background colors for each namespace. Edits made in this dialog modify the declarations of namespaces in the taxonomy.



The Namespace Prefixes dialog lists all the namespaces in the taxonomy.

- To add or delete a namespace, use the **Add** or **Delete** buttons, respectively. After adding a namespace, edit the default prefix and default URI by double-clicking in the respective field and entering the changes.
- A color can be assigned to a namespace via the color palette for that namespace. If a color has been assigned to a namespace, all components in that namespace will be displayed with this color as its background in the Main Window and entry helpers of XBRL View. Note that a color setting for a given namespace applies for that namespace across all taxonomy documents opened in XBRL View.

When you have finished editing in the Namespaces dialog, click **OK** to make your editing changes take effect.

The target namespace

The target namespace of a taxonomy is **defined** in the `xs:targetNamespace` attribute of the taxonomy's `xs:schema` element (see *listing below*). (The `xs:schema` element is the document element of the concept definitions file.)

```
<xs:schema targetNamespace="http://www.altova.com/XBRL/Taxonomies">
  ...
</xs:schema>
```

In addition to defining the target namespace (specifying it, that is), the target namespace must also be **declared** on the `xs:schema` element so that it is in scope for the entire length of the document. The listing below declares the namespace that is the target namespace.

```
<xs:schema targetNamespace="http://www.altova.com/XBRL/Taxonomies" xmlns:
ns1="http://www.altova.com/XBRL/Taxonomies" >
    ...
</xs:schema>
```

In the listing above, the namespace is declared on the `xs:schema` element and is given a prefix of `ns1`.

Setting the target namespace

When a new taxonomy is created using the New Taxonomy Wizard, a default target namespace and prefix are automatically created for the taxonomy. The default target namespace is based on data you entered in the New Taxonomy Wizard. The prefix of the default target namespace will be of the form `nX`, where `X` is an integer. The declaration of the default target namespace and prefix can then be edited by accessing the Set Target Namespace dialog (via the **XBRL | Set Target Namespace** command) and editing it there (*screenshot below*). These edits will modify not only the **definition** of the target namespace (the value of the `targetNamespace` attribute) but also the **declaration** of the target namespace.



To modify only the declaration of the target namespace (but not its definition) or the declaration of any namespace, edit the prefix and value of the namespace in the Namespace Prefixes dialog (**XBRL | Namespace Prefixes** command).

Example file: Step 2

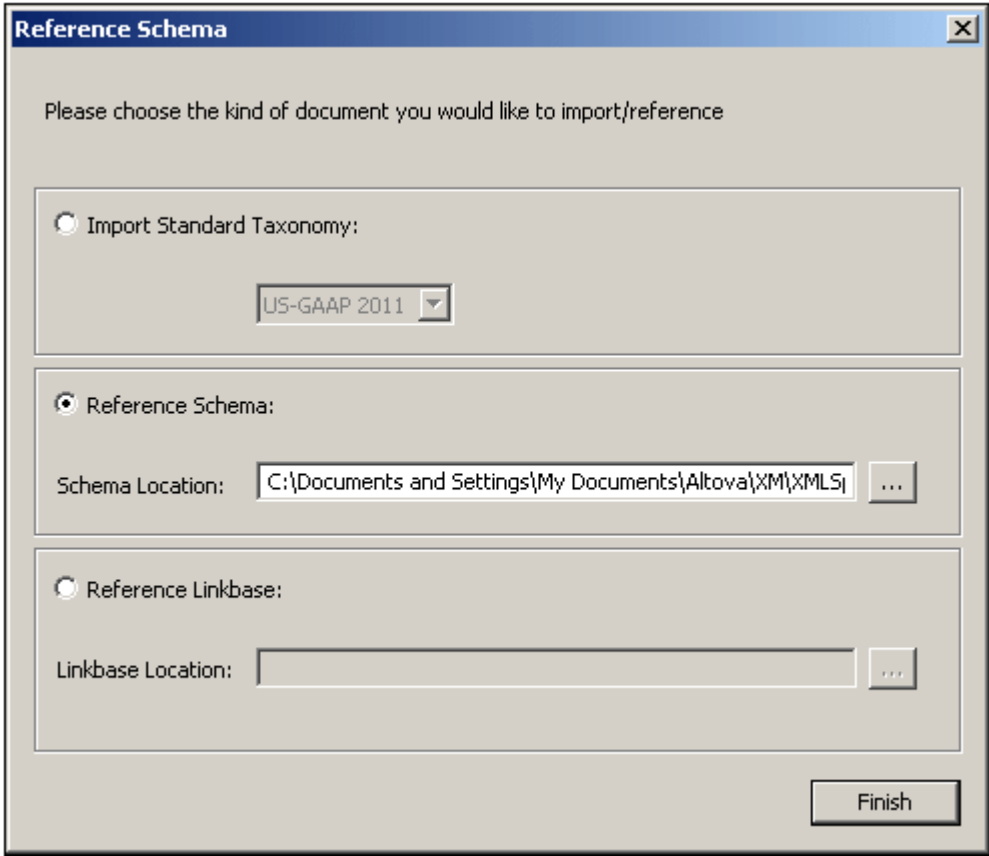
Open the Set Target Namespace dialog via the **XBRL | Set Target Namespace** command. The default target namespace is `http://xbrl.nanonull.com/2011` and it has a prefix of `n1`. Double-click the namespace and edit it as required, then do the same with the prefix. We have used the namespace `http://www.altova.com/nanonull` and assigned it a prefix of `nanonull` (see *screenshot above*). On clicking **OK** in the dialog, the target namespace will be assigned and the target namespace will be declared with the prefix you have assigned. In our case the target namespace and prefix are, respectively, `http://www.altova.com/nanonull` and `nanonull`.

In the next step we will take a closer look at the [import mechanism](#) of XBRL View.

11.5 Importing a Taxonomy

If a new taxonomy is to build upon an existing taxonomy, the existing taxonomy must be imported into the extending taxonomy. If a new taxonomy is created with the New Taxonomy Wizard, then US-GAAP-based and IFRS-based taxonomies can be imported at the time the taxonomy is created. Whether this has been done or not, a taxonomy can always be imported at a subsequent time.

To import a taxonomy, do the following: Right-click in the Overview entry helper in XBRL View and select the menu command **Import/Reference**. This pops up the Reference Schema dialog (*screenshot below*), in which you can specify a taxonomy to import or a linkbase to reference.

The image shows a Windows-style dialog box titled "Reference Schema". It contains three radio button options: "Import Standard Taxonomy:", "Reference Schema:", and "Reference Linkbase:". The "Reference Schema:" option is selected. Under "Import Standard Taxonomy:", there is a dropdown menu showing "US-GAAP 2011". Under "Reference Schema:", there is a text box for "Schema Location:" containing the path "C:\Documents and Settings\My Documents\Altova\XML\XMLS\..." and a browse button "...". Under "Reference Linkbase:", there is a text box for "Linkbase Location:" and a browse button "...". A "Finish" button is located at the bottom right of the dialog.

There are three import/reference options: (i) a standard taxonomy (US-GAAP or IFRS); (ii) any other taxonomy (reference schema); and (iii) a linkbase. If you are importing a non-standard taxonomy, select the Reference Schema radio button, click the **Browse** button of the Schema Location text box, and browse for the taxonomy you want. When you are done, click **Finish**. The selected taxonomy will be imported and its elements and relationships will be displayed in XBRL View.

Note the following:

- The Overview entry helper lists taxonomies that the imported taxonomy itself imports, as well as linkbases that the imported taxonomy uses.
- In the Global Elements entry helper, concepts defined in the imported taxonomy are listed.
- In the Design window and Details entry helper, imported concepts are indicated with a gray font color.

- You can delete an imported taxonomy by right-clicking it in the Overview entry helper and selecting **Remove**.

After you have imported a taxonomy, you can extend the taxonomy as required.

Note: If you find that a large taxonomy such as US-GAAP slows down your editing, use the [filter in the main window](#) to limit the display to elements created in the new, extending taxonomy. This will speed up editing considerably.

Import mechanism

The effect of adding a standard import as described above is to add an `xs:import` element to the new taxonomy file. The `xs:import` element specifies the namespace and location of the imported taxonomy (*listing below*).

```
<xs:import namespace="http://fasb.org/us-gaap/2011-01-31"
           schemaLocation="
http://xbrl.fasb.org/us-gaap/2011/elts/us-gaap-2011-01-31.xsd "/>
```

In the listing above, the `schemaLocation` attribute specifies that the taxonomy is to be loaded via the Internet. But this URI maps, via [XMLSpy's catalog mechanism](#), to a local copy of the US-GAAP taxonomy (that is delivered with your XMLSpy package).

To locate a locally saved taxonomy, a local address can be used directly to locate the taxonomy. Alternatively, a web address can be used which is mapped to a local address via [a catalog file](#). Accessing taxonomies from local locations will greatly speed up your work.

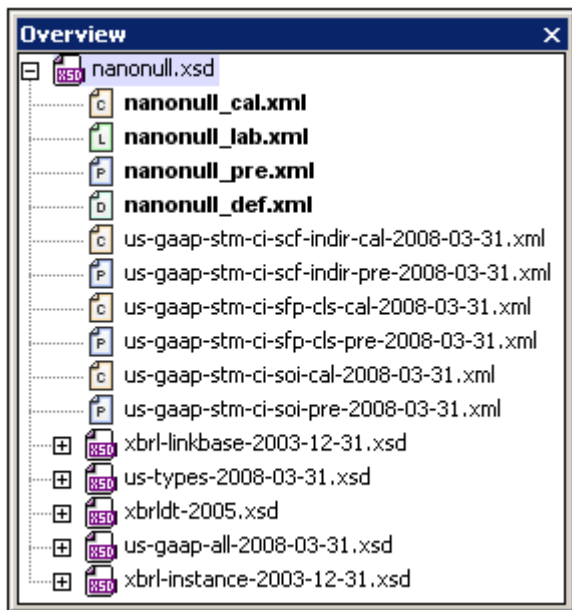
Example file: Step 3

In the example we are creating, we use the US-GAAP 2011 taxonomy. This taxonomy has already been imported during the [creation of the initial taxonomy](#). In the Overview entry helper, take a close look at all the imported taxonomies and referenced linkbases. Switch to Text View and look for the `xs:import` elements. In the Main Window, notice that imported concepts are indicated with a gray font color. Also notice that the Overview entry helper lists the linkbases and the imported schemas of the US-GAAP taxonomy.

In the [next step](#), we will take a closer look at linkbase files and the referencing mechanism.

11.6 Setting Up the Taxonomy Files

The Overview entry helper displays in a tree structure the files that constitute the taxonomy (*screenshot below*). At the root of the tree is the main taxonomy file (the concept definitions file); this is the currently active file. The files on the next level are of two types: (i) linkbase files that specify the various relationships in the taxonomy; these are indicated by [colored icons](#); and (ii) imported schemas (the .xsd files).



In the section, [Importing a Taxonomy](#), you have seen how a taxonomy can be imported via the Overview entry helper. The imported taxonomy is listed among the imported schemas in the Overview entry helper.

In this section, we show how the Overview entry helper can be used to manage linkbase files. The four operations for managing linkbases are all accessed via the Overview entry helper's context menu. They are:

- [Adding new linkbases](#) and [saving them with the taxonomy](#).
- [Setting the linkbase kind](#). In cases where the linkbase type (calculation, definition, presentation, label, or reference) is not known to XMLSpy, the linkbase type can be explicitly specified.
- Setting a linkbase as the [default linkbase](#) for a particular type of relationship linkbase. If there is more than one linkbase for a particular type of relationship, say, label relationships, then new labels that you create in the Taxonomy Editor will be created in the default label linkbase.
- [Deleting linkbases](#).

Note: There are five types of relationships: (i) definition, (ii) calculation, (iii) presentation, (iv) label, and (v) reference. Separate linkbase files can be created for each of these relationship types.

Adding a new linkbase

To add a new linkbase, do the following:

1. Right-click in the Overview entry helper and select **Add New Linkbase | <relationship type>**. A new linkbase file of the selected relationship type is created in the Overview

- entry helper with a default name. Note that the new linkbase is created as the default linkbase of its relationship type (indicated by the filename being in boldface).
2. Right-click the default name, select **Rename**, and edit the name.
 3. A newly created linkbase file is physically saved at a particular location only when the main taxonomy file is saved the next time. See below for details.

Saving linkbase files

If a linkbase file has not been saved, this is indicated by an asterisk after the name of the linkbase file. When you save the main taxonomy file, the following will happen:

1. The Confirm Linkbase Paths dialog appears. This dialog contains the names and locations (paths) of all the linkbases in the taxonomy, including the newly created linkbase files. Any unsaved linkbase file will have a default path to the folder in which the main taxonomy file will be, or has been, saved. You can edit the path of individual linkbase files if you wish to save a linkbase file to another location. You can also edit the name of the file.
2. Click **OK** when done. The linkbase files will be saved to the specified locations.

Setting linkbase kind

The linkbase kind of a file (also referred to as a file's linkbase type) can be set by using this command. Right-click the file for which the linkbase kind is to be changed, and, from the context menu, select the command **Set Linkbase Kind | <relationship type>**. The **All** option enables you to specify that the linkbase file can contain more than one kind of relationship.

Setting a default linkbase

A default linkbase file can be set for each relationship type. When a relationship of that type is defined in the Taxonomy Editor, the relationship is saved to the default linkbase file of that relationship type. To set a linkbase file as the default linkbase, right-click it and select **Set Default Linkbase**. The names of default linkbases are displayed in bold.

Deleting a linkbase

A linkbase can be removed from the taxonomy by right-clicking it and selecting **Remove**.

Example file: Step 4

The taxonomy we are creating already has linkbase files. These were created when the taxonomy was [created with the New Taxonomy Wizard](#). The original default names were changed to the following:

- Calculation linkbase: `nanonull_cal.xml`
- Definition linkbase: `nanonull_def.xml`
- Label linkbase: `nanonull_lab.xml`
- Presentation linkbase: `nanonull_pre.xml`

In order to test some of the commands introduced in this section, do the following: Create a linkbase file by using the **Add New Linkbase** command and creating any linkbase kind you like. Rename it as described above. Notice that the newly created linkbase becomes the default linkbase of its relationship type (indicated by its name being displayed in bold). Select it and set it to be some other relationship type (using the **Set Linkbase Kind** command). Notice that the file is **not** the default linkbase of its new relationship type. Now delete the linkbase (using the **Remove** command). Since one of the original linkbase files is now no longer a default linkbase, set that file as the default linkbase of its relationship type.

In the next step, we will [add new elements to the main taxonomy file](#) (or concept definitions file).

11.7 Adding Elements to a Taxonomy

To add an element (concept) to the taxonomy, click the **Add New Element** icon in the Main Window (*screenshot below*).



The new element with a substitution group of `xbrli:item` and with a default name is added to the list of elements in the display (*screenshot below*).



For a description of the element box, see [Main Window: Elements Tab](#). You can now edit the properties of the element in the [main window](#) in the following ways.

- The name of the element can be changed by double-clicking the default name and entering the correct name. Note that you must also enter the correct [namespace prefix](#) for the name.
- The substitution group of the element can be changed by expanding the element box—click the plus icon to do this—and then selecting the required substitution group from this field's dropdown list (*screenshot below*).



- To change the Balance, Period, Abstract, or Nillable property, click the corresponding icon to the left of the element name and select one of the options from the box that pops up.
- To add a label linkrole for the element, right-click anywhere in the element box and select the **Add Label Linkrole** command. A row for the label linkrole is added; in this row you can enter the label linkrole or select an option from the combo box. Note that if no label linkbase file is associated with the taxonomy, one will be created now and will be displayed in the [Overview entry helper](#).
- A label can be added for a label linkrole by right-clicking the label linkrole and selecting the **Add Label** command. To enter the details of the label, either double-click in the field to be edited and enter the new value, or select the new value from the respective combo boxes. The changes you make to labels will be saved to the label linkbase when the main taxonomy file is saved.
- References are added to the reference linkbase in the same way that labels are added to the label linkbase. First, a reference linkrole is added for the element, then a reference is added for a specific reference linkrole.

Element properties can also be edited in the Details entry helper. See [Entry Helpers in XBRL View](#) for a description of how to do this.

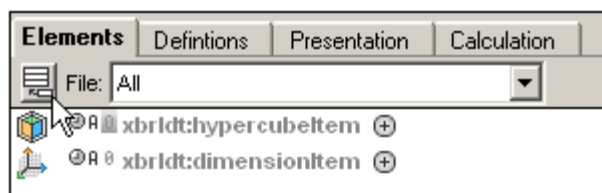
Example file: Step 5

In this section, we will extend the US-GAAP taxonomy by creating new elements.

The first element we will create is the item `nanonull:OnboardAndOther`, which represents revenues from the sale of items on board Nanonull's cruise ships. This specific revenue head is not available in the US-GAAP taxonomy, which is why it must now be created as an extension of US-GAAP. As a new element created specially for the Nanonull taxonomy, it must be created in the Nanonull namespace (<http://www.altova.com/nanonull>), which has been declared with a prefix of `nanonull`. Creating the element with this prefix will put this element in the Nanonull namespace.

To create the element, do the following:

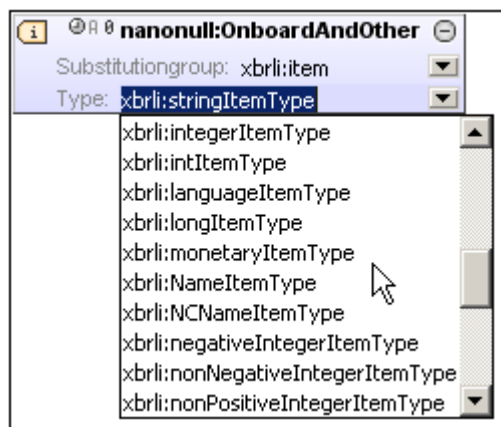
1. Click the **Add New Element** icon in the Main Window (*screenshot below*).



A new element with a substitution group of `xbrli:item` and with a default name is added to the list of elements in the display (*screenshot below*).



2. Double-click the element name and enter the name `nanonull:OnboardAndOther` (*screenshot below*). This creates the element `OnboardAndOther` in the Nanonull namespace.
3. Expand the element box and, since the element will contain a monetary amount, change the `Type` attribute to `xbrli:monetaryItemType` (*screenshot below*).



- Now click to the left of the clock icon and, from the popup that appears, select `credit` (screenshot below).



This sets the value of the `xbrli:balance` attribute to `credit`.

- Click on the clock, A, and 0 icons, and set the values of the `xbrli:duration`, `xs:abstract`, and `xs:nillable` attributes to *duration*, *NOT Abstract*, *Nillable*, respectively. (In the `.xsd` file, the actual attribute values will be: `credit`, `duration`, `false`, and `true`, respectively.)
- Right-click the element box and, from the menu that pops up, select **Add Label Linkrole**. This creates a label linkrole row at the bottom of the element box (screenshot below).



- Select the XBRL link URI.
- Right-click the label linkrole row and from the menu that pops up select **Add Label**. This creates a label row within the label linkrole.
- Double-click in the language field of the newly created label row (screenshot below) and enter `en-us`; in the next field which is the linkrole field, select the `documentation` role from the dropdown list; in the label field, enter the text that should appear in documentation. Then create another label row for the `label` linkrole by repeating Step 9. The value of the `label` role will appear in the Taxonomy Editor in the Main Window and in entry helpers when the label display is switched on (in preference to short or long names).



The element `nanonull:OnboardAndOther` has now been successfully created.

Notice that `OnboardAndOther` had an `xbrli:balance` value of `credit`. This is because it is a revenue item: money is coming in. Since the items being sold on board will have costs attached to them, i.e. cost the company to procure, we will also create a debit-side element called `nanonull:CostOfOnboardAndOther`. Create this element the same way as `nanonull:OnboardAndOther` was created, with one difference, however: set the value of `xbrli:balance` to `debit` instead of `credit`.

Another cost to be included is for commissions to agents. This should be taken care of with a debit element called `nanonull:CruiseCommissionsTransportationAndOther`. Create this element exactly as you did `nanonull:CostOfOnboardAndOther`.

Finally, we add three abstract elements, `Asia`, `Europe`, and `US`, so that concepts can be grouped by region. Since the elements are used only for grouping purposes and will not themselves have values, they are known as abstract elements. What type such an element has

is therefore immaterial. It is best to give an abstract element a type that matches its semantics. For example, we have given the abstract elements `Asia`, `Europe`, and `US`, a type of `stringItemType`. Create the `nanonull:Asia`, `nanonull:Europe`, and `nanonull:USA` elements just as you created the previous elements. The only difference this time will be that the value of the `Abstract` attribute must be set to *Abstract* (actual attribute value in the XSD file will be `true`) and there will be no `xbrli:balance` attribute.

Note: If an `xbrli:balance` attribute is present on an abstract element, this abstract element must be of type `monetaryItemType`, otherwise the taxonomy will be invalid. It is best to omit the optional `xbrli:balance` attribute from all abstract elements.

In the next step we will [specify linkroles](#) for the new taxonomy. These linkroles will be needed when we create new relationships.

11.8 Relationships and Linkroles

When a set of relationships is created these relationships are created within a containing element. For example, when definition relationships are created, the elements defining the definition relationships (the locators and definition arcs) are all created within a `definitionLink` element, which looks something like this:

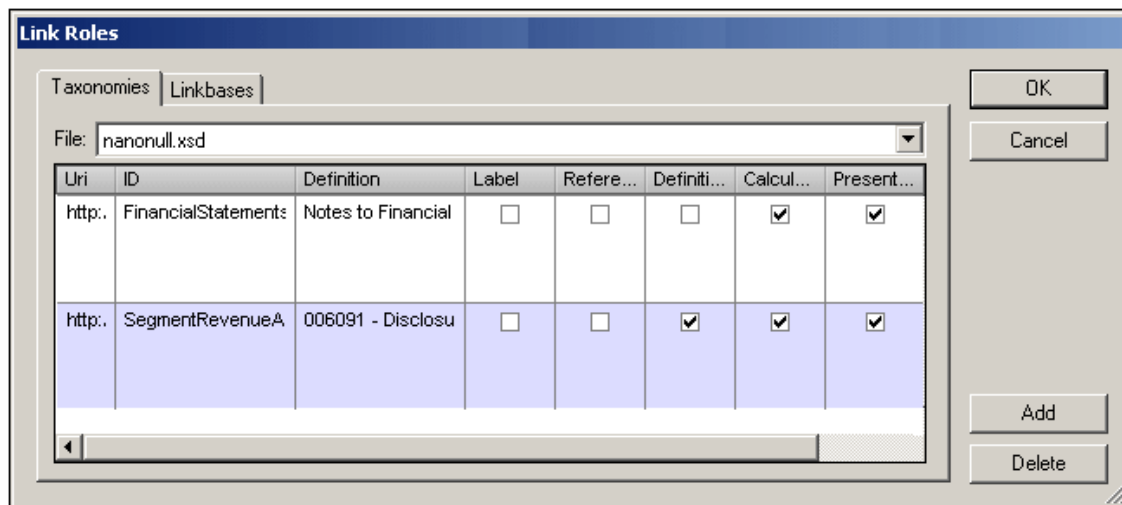
```
<link: definitionLink xlink:type="extended"
xlink:role="
http://www.nanonull.com/taxonomy/role/SegmentRevenueAndOperatingIncome">
```

The value of the `xlink:role` attribute in the definition link (as in the definition link listed above) must be the value of the `roleURI` attribute of one of the linkroles set to be used on definition relationships (see listing below). A linkrole (as in the listing below) is contained in the `appinfo` element of the taxonomy.

```
<xs:appinfo>
  <link:roleType id="SegmentRevenueAndOperatingIncome"
    roleURI="
http://www.nanonull.com/taxonomy/role/SegmentRevenueAndOperatingIncome">
    <link:definition>006091 - Disclosure - Segment Revenue and Operating
Income</link:definition>
    <link:usedOn>link:calculationLink</link:usedOn>
    <link:usedOn>link:definitionLink</link:usedOn>
    <link:usedOn>link:presentationLink</link:usedOn>
  </link:roleType>
</xs:appinfo>
```

A linkrole can be used in the containing elements of other relationship kinds besides in `definitionLink` elements (for example, in `calculationLink` and `presentationLink` elements). In the listing above, notice that there are `usedOn` elements that specify in which kind of relationships this linkrole may be used.

To create linkroles in a concept definitions file (main taxonomy file), in XBRL View, click the menu command **XBRL | Linkroles**. This pops up the Link Roles dialog (screenshot below).



In the Taxonomies tab, select the taxonomy and click **Add** to add a linkrole. Then specify the linkrole's URI and ID (refer to listing above). Now specify for which kinds of relationships this linkrole should be available; do this by checking the check boxes of the required relationship kinds (see screenshot above).

Example file: Step 6

Create two linkroles via the Link Roles dialog (**XBRL | Linkroles**) as described above and shown in the screenshot above:

1. id="SegmentRevenueAndOperatingIncome"
URI="http://www.nanonull.com/taxonomy/role/SegmentRevenueAndOperatingIncome" (to be used on definition, calculation, and presentation relationships)
2. id="FinancialStatements"
URI="http://www.nanonull.com/taxonomy/role/FinancialStatements" (to be used on calculation and presentation relationships)

In the next step we will [create relationships](#) for the new taxonomy.

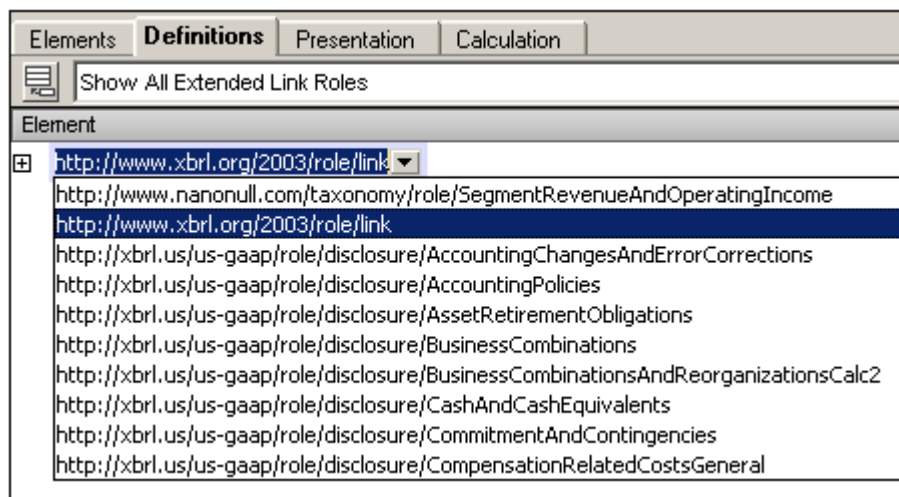
11.9 Creating Relationships: Part 1

Relationships are created in their respective tabs: Definitions, Presentation, Calculation. The way all three kinds of relationships are created is similar, with the biggest difference being that definition relationships have arcoles, while presentation relationships and calculation relationships do not have arcoles. In this section we describe how to create relationships using definition relationships. In the [next section](#) we explain how presentation and calculation relationships are different, as well as other features relating to relationships.

While reading the description below, we recommend that you open a finished taxonomy in XBRL View. You can find the Nanonull taxonomy (`nanonull.xsd`) in the folder `C:\Documents and Settings\<username>\My Documents\Altova\XMLSpy2012\Examples\XBRLExamples\Nanonull`.

Adding the linkrole

Click the required relationships tab in the Main Window (Definitions, Presentation, Calculation). Then right-click in the Main Window and select the **Add Extended Link Role** command. This adds a line containing the URI of a default linkrole (*screenshot below*). Click the dropdown arrow at the right-hand side of this line to display a list of available linkroles and select the required linkrole.

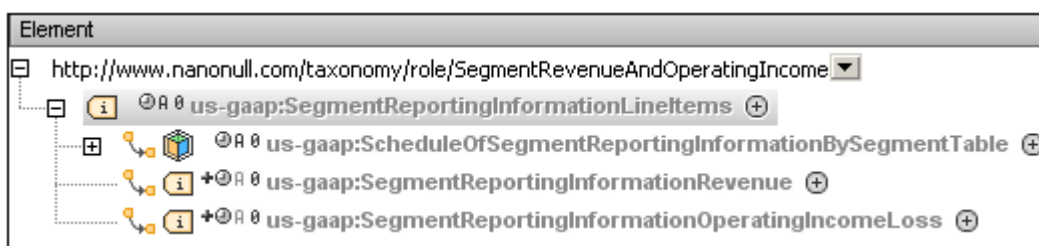


If the required linkrole is not available, this is because it has not been defined either in the taxonomy or for the current relationship kind. See [Relationships and Linkroles](#) for details about linkroles and how to create them.

Any number of linkroles can be added.

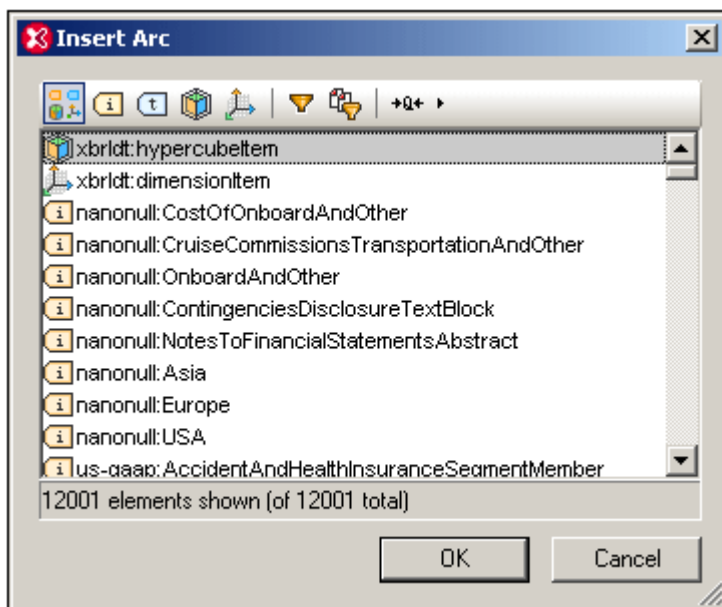
Inserting element references and arcs within a linkrole

The first element to create within a linkrole is one **from** which a relationship will be created to another element (see *screenshot below*). This will usually be an abstract element that groups other elements under it (for example, an element for a balance sheet). This element will have no entry in the arcoles column because it is at the **from** end of an arc. Arcoles are listed on the elements at the **to** end of an arc.

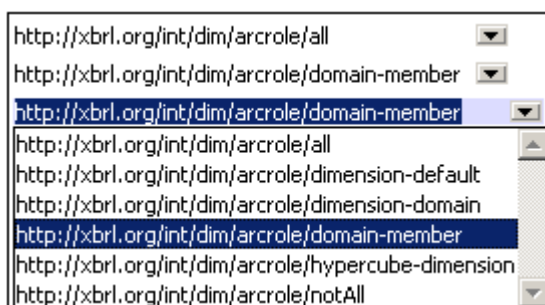


In the screenshot above the highlighted element is the inserted element reference. It has three arcs, one to a hypercube element and two to item elements. These three elements are at the **to** end of their respective arcs and the **from-to** relationship is defined by the corresponding arcoles, which are displayed in the Arcrole column.

To insert an arc on an element reference or element, right-click the **from** element and select **Insert Arc** from the context menu that pops up. This causes the Insert Arc dialog (*screenshot below*) to appear. Select the element to be created at the **to** end of the arc. To filter the view in this dialog, switch on the filter and specify a condition for the filter (see [Entry Helpers in XBRL View](#) for a description of how to do this).



The element will be inserted with a default arcrole. You can change the arcrole by selecting an alternative from the dropdown list of the arcrole (*screenshot below*).

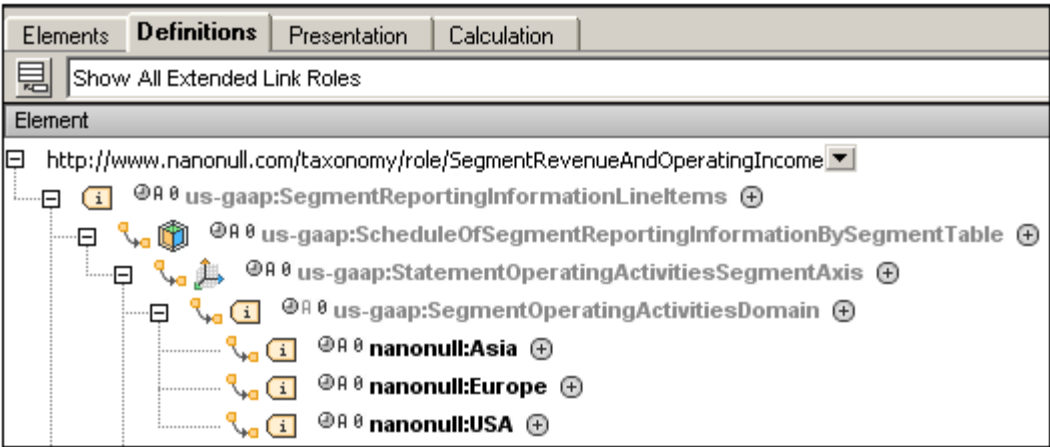


Note: Elements, with arcs, can also be added by dragging them from the Global Elements

entry helper.

Example file: Step 7

Create definition relationships as shown in the screenshots below using the method described above.



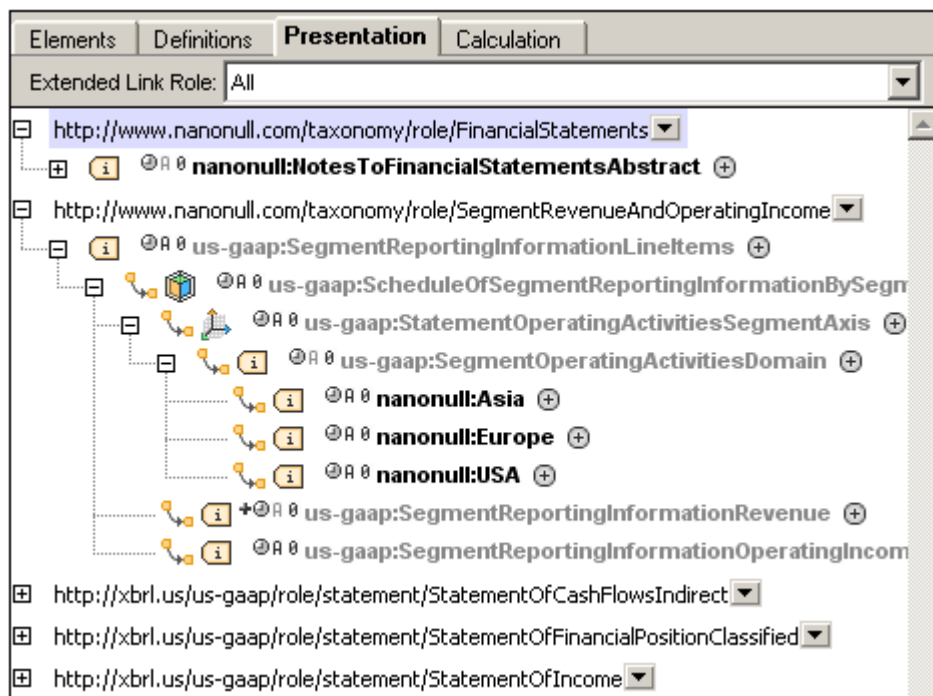
The screenshot above shows the elements to add with arcs. The screenshot below shows the arcoles of the newly added elements.



You can compare the taxonomy you have created with that supplied with your XMLSpy package. The supplied taxonomy (nanonull.xsd) is in the folder C:\Documents and Settings\<username>\My Documents\Altova\XMLSpy2012\Examples\XBRLExamples\Nanonull.

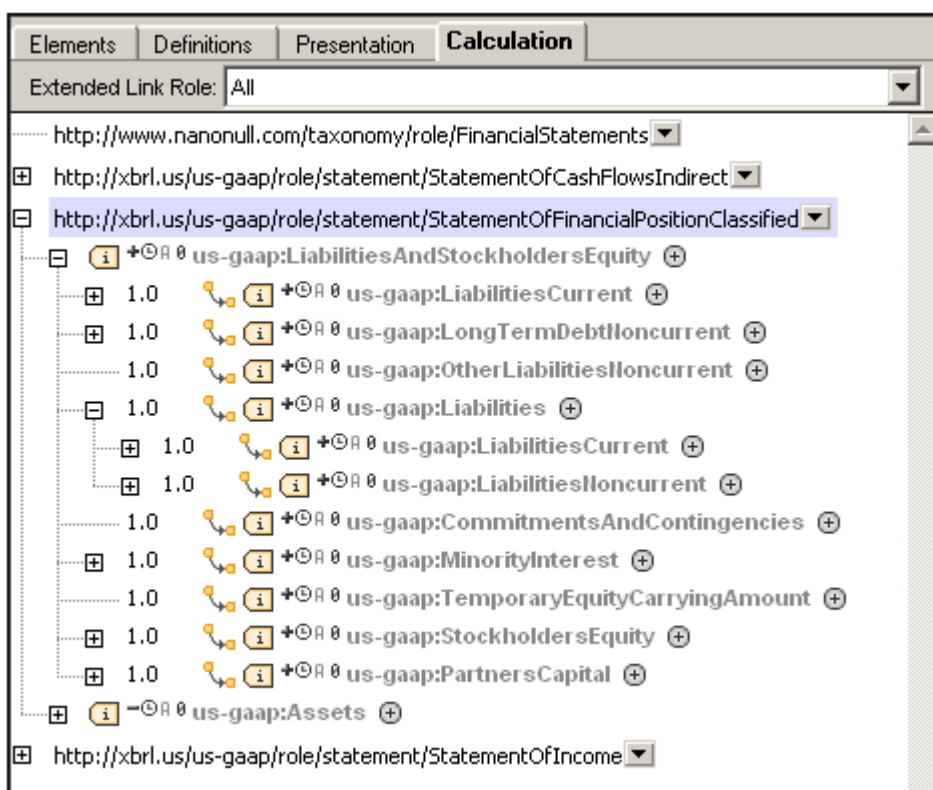
11.10 Creating Relationships: Part 2

The previous section, [Creating Relationships: Part 1](#), explained how to create relationships using definition relationships to demonstrate the mechanism. Presentation relationships (*screenshot below*) and calculation relationships are created in a similar way. The only difference is that there is no Arcrole column in presentation and calculation relationships.



The following points should be noted:

- Presentation and calculation relationships can be considered to be a simple arc between two elements in the manner of parent-child relationships. The arc icons signify this relationship. So inserting an arc on an element is equivalent to creating a child element in the graphical representation. Using arcs, therefore, a hierarchy can be built up.
- Elements can also be dragged from the Global Elements entry helper into the tree. These elements are always dropped at the **to** position of an arc. An arrow appears when the element is in position to be dropped.
- Calculation arcs have **weight** attributes that indicate how the value of the **to** element in the arc should be summed (see *screenshot below*). For example, a weight value of **+1.0** indicates that 100% of the element's value should be added towards the value of the **from** (or summation) element. A value of **-1.0** indicates that 100% of the value of should be subtracted from the value of the summation element. Double-clicking the **weight** attribute value enables you to enter an optional value.



The `weight` attribute can also be modified in the Details entry helper (see *below*).

Prohibiting the use of an arc

All arcs, whether definition, presentation, or calculation, have a `use` attribute that can take a value of `optional` or `prohibited`. When the value `prohibited` is used, the arc is negated.

Color and context menu

When elements have been created in the current taxonomy and can be edited, they are displayed in black. Otherwise (when they are from imported taxonomies, which must not be edited) elements are displayed in gray.

The following entries appear in context menus in the main window of the relationships tabs.

- *Insert element reference*: Available on extended linkroles. Adds an element under the linkrole that will always be at the `from` end of arcs.
- *Delete element reference*: Available on element references immediately under a linkrole.
- *Insert arc*: Available on elements. Inserts an arc and pops up a dialog in which the element to be at the `to` end of the arc can be selected.
- *Set target role*: Sets a target role on the selected element.
- *Add label linkrole*: Adds a label linkrole to the selected element.
- *Add reference linkrole*: Adds a reference linkrole to the selected element.
- *Override arc*: Replaces the (implicit) `optional` value of the `use` attribute of the arc with the `prohibited` value, thus negating the arc.
- *Remove arc*: Removes the selected arc.
- *Show in global elements*: Highlights the selected element in the Global Elements entry helper.

Details entry helper

When an element in a relationship is selected, arc attributes can be edited in the Details entry helper (*screenshot below*).

The screenshot shows a window titled "Details" with a close button (X) in the top right corner. The window displays information for the element "us-gaap:ConvertibleDebtCurrent". The interface is organized into sections: "General", "Arc", and "Children".

us-gaap:ConvertibleDebtCurrent	
▽ General	
Defined in	http://xbrl.us/us-gaap/1.0/elts/us-gaap-2008-03-31.xsd
Namespace	http://xbrl.us/us-gaap/2008-03-31
Abstract	☐
Nilable	☒
Balance	credit
Period Type	instant
Subst. Group	xbrli:item
Type	xbrli:monetaryItemType
▽ Arc	
Order	20.0
Priority	36671884
Target Role	
Arc Role	http://www.xbrl.org/2003/arcrole/summation-item
Use	prohibited
Weight	1.0
Context Element	
Usable	
Typed Domain Ref	
▽ Children	
Arcs	0
Label Children	1
Reference Children	1

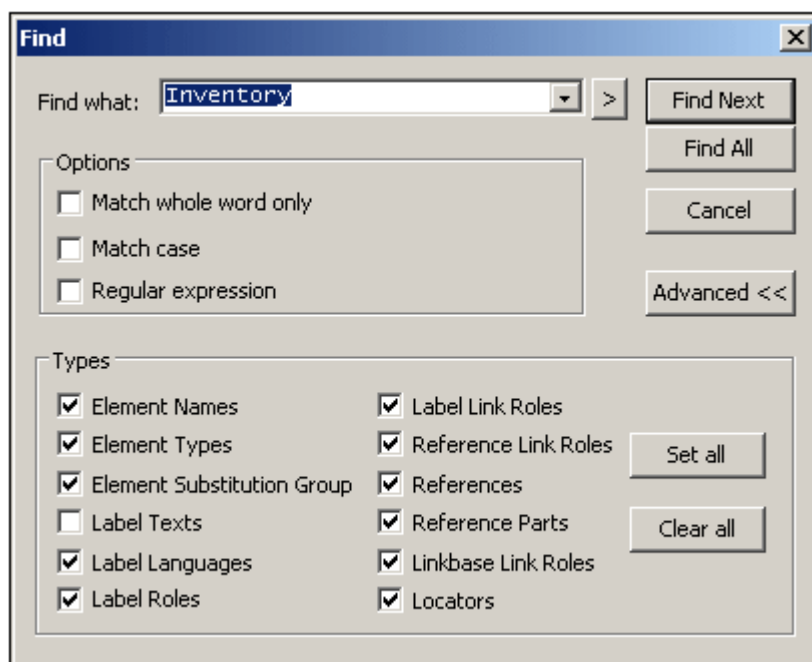
Attributes which cannot be edited in the graphical display in the main window—such as `order` and `priority`—can be edited in the Details entry helper.

11.11 Find in XBRL

In XBRL View, XBRL taxonomies can be searched using XMLSpy's Find in XBRL feature, which is enabled when an XBRL taxonomy is active in XBRL View. The Find in XBRL feature is accessed in one of the following ways:

- Via the **Edit | Find** menu command when an XBRL taxonomy is active in XBRL View.
- Via the **Find** button in the Find in XBRL window.
- By pressing **Ctrl+F**.

Selecting any of these access methods pops up the Find dialog (*screenshot below*).

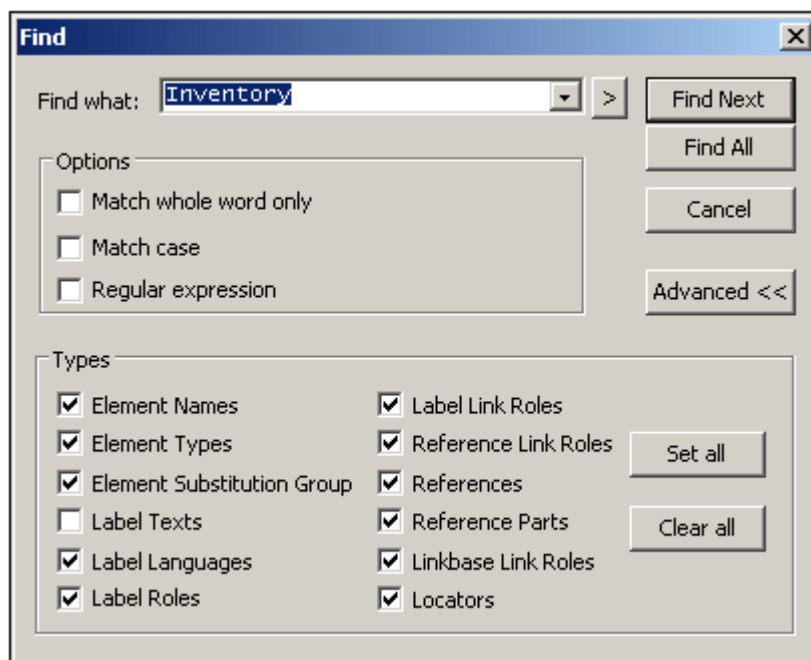


Usage is as follows:

- [Enter the search term](#) in the *Find what* text field of the Find dialog (*screenshot above*) and check the required options
- [Specify the XBRL component types to be searched](#) in the Types pane
- [Execute the command](#) using the **Find Next** or **Find All** button
- [Use the Find in Schemas window](#) to view the search results and navigate to a component quickly

11.11.1 Search Term

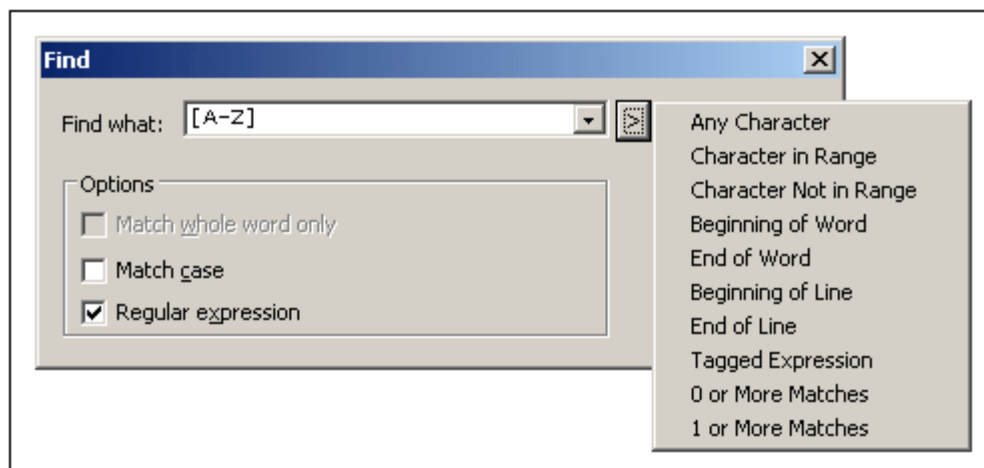
A search term can be specified to be case-sensitive or to match a whole word by checking the respective options in the Options pane (*see screenshot below*). If you wish to search using a regular expression, check the *Regular Expression* option in the Options pane before clicking the **Find Next** or **Find All** button. See below for more details about using regular expressions.



Note: A whole word is considered to be delimited by any character that is not alphanumeric or the underscore character. So the search term `asset` will return the text `xbrl: asset`, since the colon character (`:`) is considered to be a word delimiter.

Regular expressions

You can use regular expressions to further refine your search criteria. A pop-up list is available to help you build regular expressions. To access this list, click the `>` button to the right of the input field for the search term.



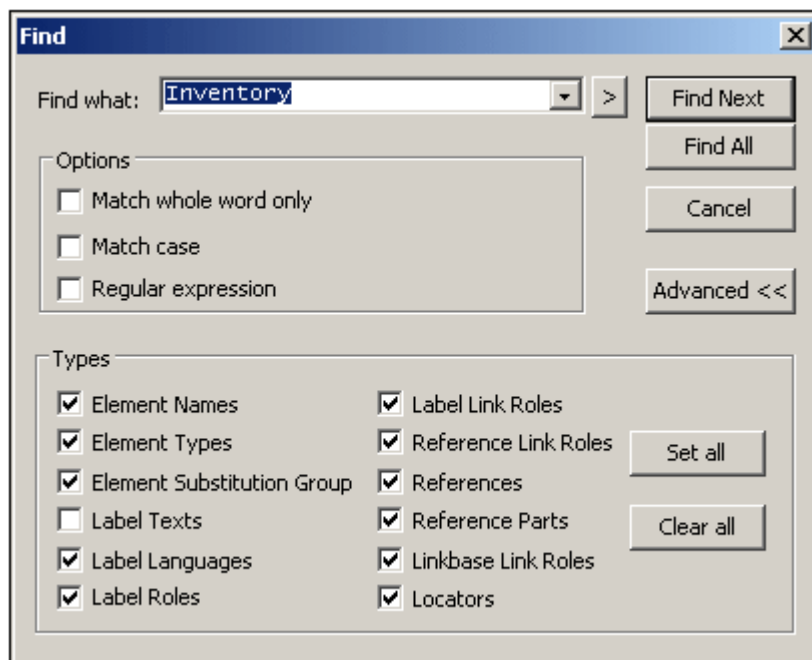
Clicking on the required expression description inserts the corresponding expression syntax character in the input field. Given below is a list of regular expression syntax characters.

- `.` Matches any character. This is a placeholder for a single character.
- `\(` Marks the start of a region for tagging a match.
- `\)` Marks the end of a tagged region.

\<	Matches the start of a word.
\>	Matches the end of a word.
\x	Allows you to use a character x, that would otherwise have a special meaning. For example, \[would be interpreted as [and not as the start of a character set.
[...]	Indicates a set of characters, for example, [abc] means any of the characters a, b or c. You can also use ranges, for example [a-z] for any lower case character.
[^...]	The complement of the characters in the set. For example, [^A-Za-z] means any character except an alphabetic character.
^	Matches the start of a line (unless used inside a set, see above).
\$	Matches the end of a line. Example: A+\$ to find one or more A's at end of line.
*	Matches 0 or more times. For example, Sa*m matches Sm, Sam, Saam, Saaam and so on.
+	Matches 1 or more times. For example, Sa+m matches Sam, Saam, Saaam and so on.

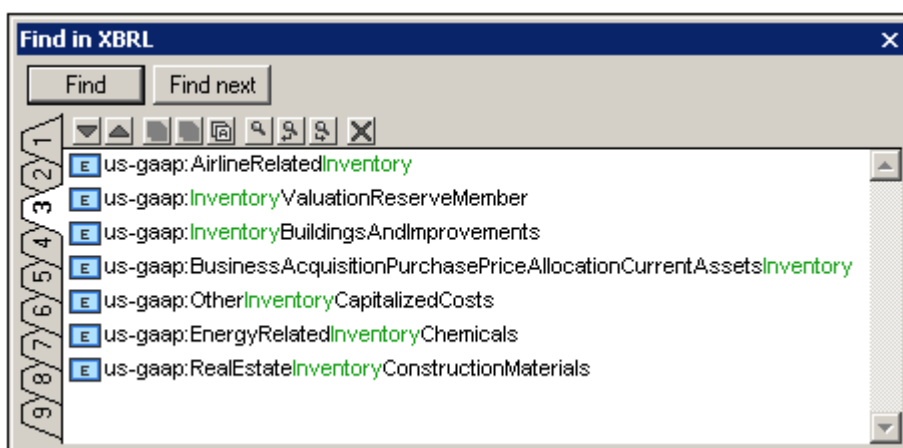
11.11.2 Command Execution

After the search term has been entered, and the options and filter for component types have been set, there are two command execution options: **Find Next** and **Find All**. These are available as buttons in the Find dialog (see *screenshot below*).



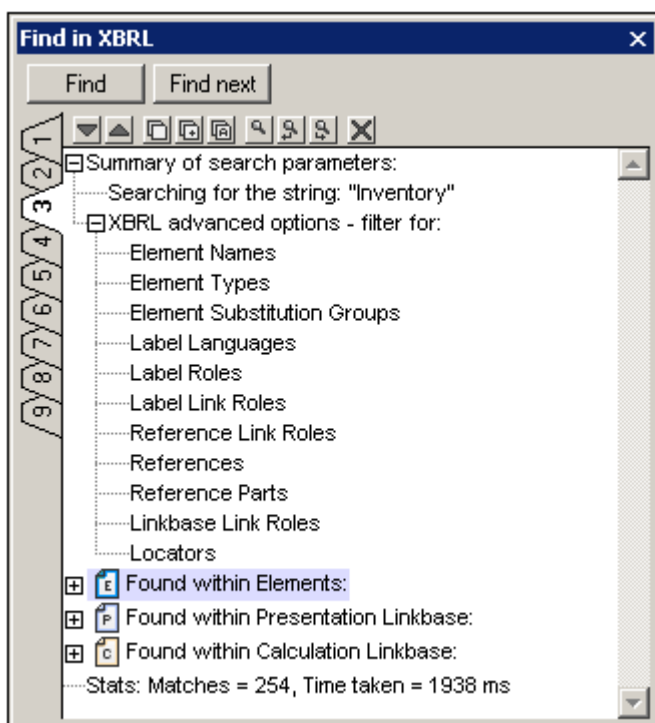
Find Next

The **Find Next** command displays, in the Find in XBRL window (*screenshot below*), the next instance of the search term. The search for the next instance will start at the next cell from the current cursor position in the active document. The **Find Next** process can be continued till all instances in the document are displayed.



Find All

The **Find All** command displays all instances of the search term, together with a summary of the search, in the Find in XBRL window (screenshot below).

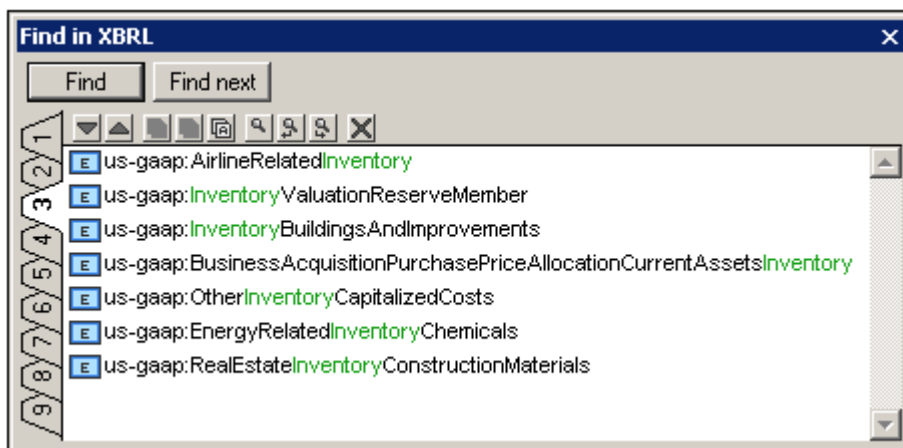


The result provides: (i) a summary of the XBRL component types that were searched; (ii) a list of the found instances of the search term, ordered according to linkbase; and (iii) statistics about the search, including the matches found and the time taken for the search. Each linkbase group can be expanded or collapsed to view the matches in that group. Clicking a match highlights the corresponding element in the XBRL document in the Main Window.

For information about the features of the Find in XBRL window, see the section [Results and Information](#).

11.11.3 Results and Information

Each time a **Find** or **Find Next** command is executed the results of the command execution are displayed in the Find In XBRL window (*screenshot below*). The term that was searched for is displayed in green; (in the screenshot below, it can be seen that `Inventory` was the search term).



Features of the Find In XBRL window

Results are displayed in nine separate tabs (numbered 1 to 9). So you can keep the results of one search in one tab, do a new search, and compare results. Clicking on a result in the Find In XBRL window pops up and highlights the relevant component in the Main Window of XBRL View. In this way, using the Find in XBRL window you can search and navigate quickly to the desired component.

The following Find In XBRL toolbar commands are available:

- The **Next** and **Previous** icons select, respectively, the next and previous find results to the currently selected result.
- The **Copy Messages** commands copy, respectively, the selected message, the selected message and its children messages, and all messages, to the clipboard.
- The **Find** commands find text strings in the Find In XBRL window.
- The **Clear** command deletes all messages in the currently active tab.

12 Office Open XML, ZIP, EPUB

Office Open XML (OOXML), ZIP files, and EPUB files are similar in that both are packages containing other files. XMLSpy's Archive View provides an interface that enables you to view the internal structure of these packages, modify these structures, and access the files in the package for editing in XMLSpy. In the case of EPUB files, Archive View also enables you to directly view the EPUB book in the Browser View of XMLSpy.

Office Open XML (OOXML)

OOXML is a file format for describing documents, spreadsheets, and presentations. It was originally developed by Microsoft for the company's Office suite of products but is now an open ECMA specification.

Structure of an OOXML file

Each OOXML document is a package of multiple files that follows the Open Packaging Convention. A package consists of XML and other data files (such as image files) plus a relationships file that specifies the relationships among the various files in the package.

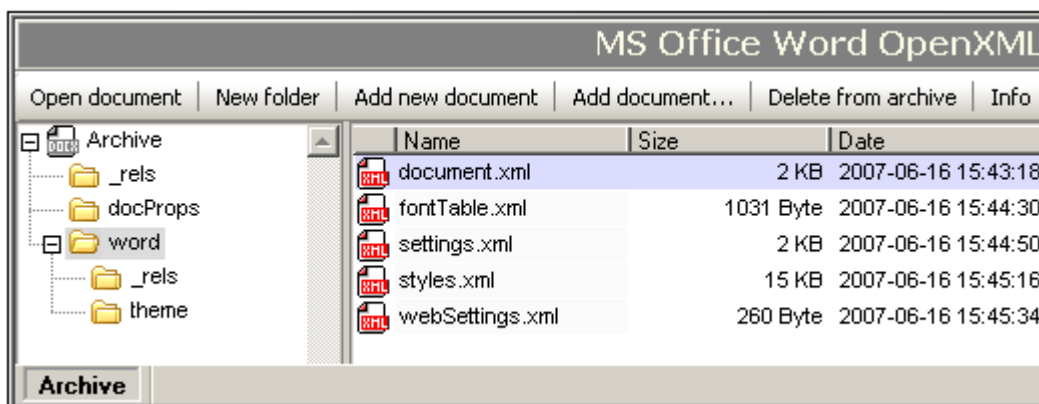
The internal structure and internal folder and file names of an OOXML file vary according to the document type. However, there is a common basic structure: an XML file called `[Content_Types].xml` at the root of the directory structure, and three directories: `_rels`, `docProps`, and a directory specific to the document type (in the case of `.docx` documents, for example, this folder would be called `word`; `xl` in `.xlsx` documents, and `ppt` in `.pptx` documents).

```
OOXML File
|-- File:      [Content_Types].xml
|-- Folder:    _rels
|-- Folder:    docProps
|-- Folder:    word/xl/ppt
```

- The `_rels` folder contains a `rels.xml` file, which specifies the relationships between the various files in the package.
- The `docProps` folder contains `app.xml` and `core.xml`, which describe key document properties.
- The `word`, `xl`, and `ppt` folders contain XML files that hold the content of the document. For example, in the `word` folder, the file `document.xml` contains the core content of the document.

OOXML in XMLSpy's Archive View

In XMLSpy's Archive View (*screenshot below*), you can view and edit the contents of an OOXML file.



Folder View on the left-hand side shows the folders in the package, whereas the Main Window shows the files in the folder selected in Folder View. In Archive View, files and folders can be added to and deleted from the archive. Also, files can be opened quickly for editing in XMLSpy by double-clicking the file in Archive View.

Intelligent editing of OOXML's internal files

The XML documents within OOXML packages are based on standard schemas. XMLSpy provides intelligent editing support for OOXML documents, in the form of entry helpers, auto-completion, and validation.

ZIP files

ZIP files archive multiple files in a lossless data compression package. These files can be of various types. In XMLSpy's Archive View, ZIP files can be created, the internal structure modified, and files in the archive edited. These operations are described in the [ZIP Files](#) subsection of this section.

EPUB files

An EPUB file is a zipped group of files used for the distribution of digital publications (EPUB books). In [Archive View](#), you can open EPUB files, create and edit EPUB files, preview the digital EPUB book, edit component files of the EPUB archive directly in XMLSpy, validate the EPUB file, and save the component files back to the EPUB archive. See the section, [EPUB Files](#), for details.

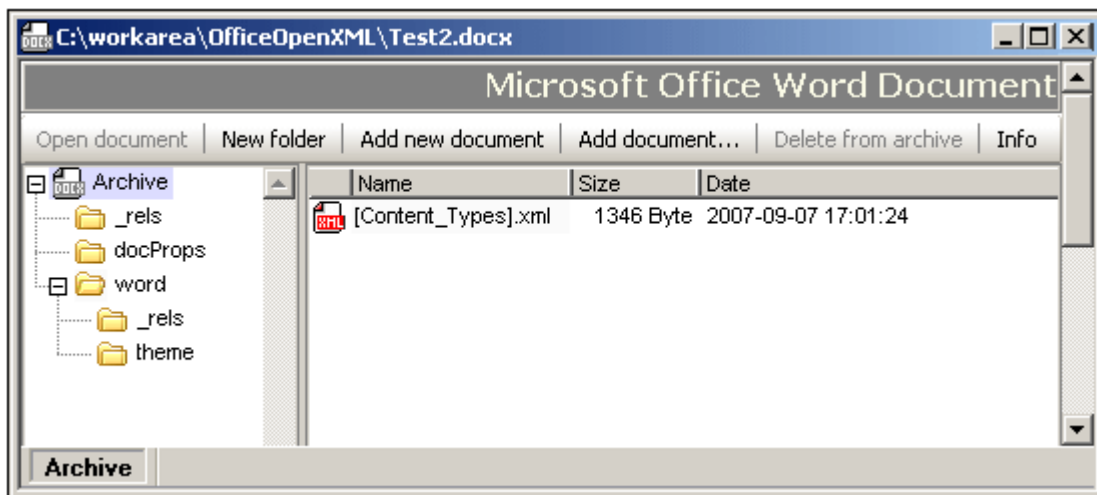
12.1 Working with OOXML Files

This section describes how to work with OOXML documents in Archive View. The following procedures are discussed:

- [Creating, opening, and saving OOXML files](#)
- [Editing the structure of an OOXML file](#)
- [Opening, editing, and saving internal OOXML documents](#)
- [Intelligent editing of internal OOXML documents](#)
- [Addressing documents in OOXML files](#)
- [Comparing OOXML archives](#)

Creating, opening, and saving OOXML files

OOXML files are created via the Create New Document dialog (**File | New** command), in which you select the required file type (. docx, . pptx, or . xlsx). You are prompted for a file name and a location at which to save the file. The new file is created at the specified location and then opened in Archive View (*screenshot below*). Notice that the basic internal structure of the OOXML document has been created.



An existing OOXML file is opened in Archive View via the Open dialog (**File | Open**) of XMLSpy. OOXML files are saved with the **File | Save (Ctrl+S)** command. This command saves the structure and relationships of the OOXML file.

Editing the structure of an OOXML file

The contents of an OOXML file can be modified by adding and deleting folders and documents to it using [Archive View](#) functionality. After these structural changes have been made, the OOXML file must be saved (**File | Save**) for the modifications to take effect. You should note the following points:

- When a new folder or document is added using the [command buttons in Archive View](#), it should be named immediately on its being created. It is not possible to rename a folder or document in Archive View.
- After a new document has been added to an archive folder, it is saved to the archive by saving it in its own window or by saving the OOXML file.

Opening, editing, saving internal OOXML documents

An internal OOXML document—that is, a document within an OOXML file package—is opened from Archive View by double-clicking it, or by selecting it in the Main Window and clicking the [Open document](#) command button. The document opens in a separate XMLSpy window. After editing it, simply save the document to save it back to the OOXML archive; there is no need to save the OOXML file itself.

Intelligent editing of internal OOXML documents

XMLSpy provides intelligent editing features for internal Office Open XML documents—that is, for documents within an OOXML file package. These features include entry helpers, auto-completion, and validation.

Addressing documents in OOXML files

Documents in OOXML files can be addressed using normal file paths plus the pipe character. For example, the file path:

```
C:\Documents and Settings\<username>\My Documents\Altova\XMLSpy2012\
\Examples\Office20XX\ExcelDemo.xlsx|zip\xl\tables\table1.xml
```

locates the file `table1.xml`, which is in the `xl\tables` folder of the OOXML file `ExcelDemo.xlsx` located in the `Examples\Office20XX` folder of the XMLSpy examples folder.

Comparing OOXML archives

When an OOXML file is open in Archive View, you can compare it with another archive by using the command [Tools | Compare Directories](#).

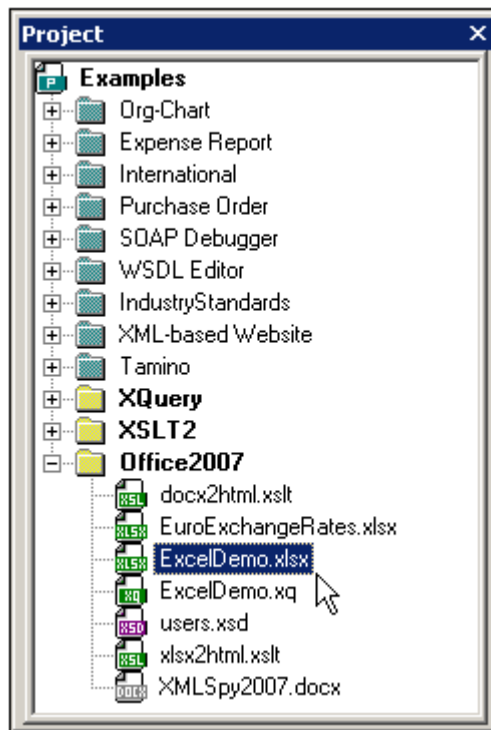
12.2 OOXML Example Files

In the `Examples\Office2007` folder of your XMLSpy application folder are the following example files:

- OOXML files: (i) a Word Open XML file (.docx), (ii) an Excel Open XML file (.xlsx), and (iii) a PowerPoint Open XML file (.pptx)
- XSLT files: (i) `docx2html.xslt` (to convert the sample .docx file to HTML), (ii) `xlsx2html.xslt` (to convert the sample .xlsx file to HTML), and (iii) `pptx2html.xslt` (to convert the sample .pptx file to HTML)
- An XQuery file: `ExcelDemo.xq` (to retrieve data from the .xlsx file)

The XSLT and XQuery files are intended to demonstrate how XSLT and XQ can be used to access and transform data in OOXML files. To run the XSLT and XQuery documents, you can use any of the following options:

- Open the OOXML file in Archive View. In Folder View, select `Archive` and then click the menu command **XSL/XQuery | XSL Transformation** (for an XSLT transformation) or **XSL/XQuery | XQuery Execution** (for an XQuery execution). Browse for the XSLT or XQuery file and click **OK**.
- In the Project Window of XMLSpy, right-click the .xlsx or .docx file in the `Office2007` folder of the `Examples` project (*screenshot below*), and select the transformation command. Browse for the transformation file and click **OK**.



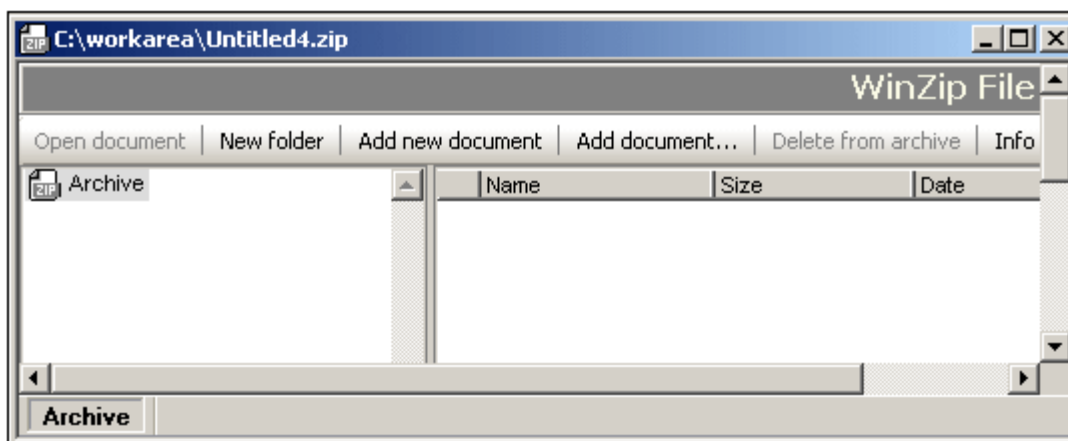
- Open the XSLT or XQuery file in XMLSpy and click the menu command **XSL/XQuery | XSL Transformation** and **XSL/XQuery | XQuery Execution**, respectively. When prompted for the XML file to transform, browse for the .docx, .xlsx, or .pptx file (according to whether the XSLT/XQ document is intended for Word or Excel).
- You can also run these operations using AltovaXML, which is a standalone package containing the Altova XSLT and XQuery Engines.

12.3 ZIP Files

In [Archive View](#), you can create WinZip files, modify the internal structure of ZIP files (WinZip, WinRAR, etc), and edit files in the ZIP package directly in XMLSpy and save the files back to the ZIP archive.

Creating and saving a WinZip file

A WinZip file is created via the Create New Document dialog (**File | New** command), in which you select the file type `.zip`. An empty WinZip archive is created in a new window in XMLSpy (*screenshot below*). You must now save the ZIP file to the desired location with the **File | Save (Ctrl+S)** command. Add folders and files as described below, and then save the ZIP file to save your additions and changes.



An existing ZIP file is opened in Archive View via the Open dialog (**File | Open**) of XMLSpy.

Note: Creating a new ZIP file is different than creating a new OOXML file in that you are not prompted for a location to save the file before the archive is opened in Archive View. For the ZIP file to be saved from the empty archive that is opened in Archive View, you must explicitly use the **File | Save (Ctrl+S)** command..

Adding folders and files and modifying the archive structure

You can add folders (click the **New Folder** button), existing files (**Add Document**), and new files (**Add New Document**) to the selected Archive folder. Note that when you add a new folder or new document, you must immediately enter a name for the folder or file; it is not possible to rename folders in Archive View.

Addressing documents in ZIP files

Documents in ZIP files can be addressed using normal file paths plus the pipe character. For example, the file path:

```
C:\Documents and Settings\<username>\My Documents\Altova\XMLSpy2012\
\Examples\Test.zip|zip\TestFolder\MyFile.xml
```

locates the file `MyFile.xml`, which is in the `TestFolder` folder of the ZIP file `Test.zip` located in the `Examples` folder of the XMLSpy examples folder.

Comparing ZIP archives

When a ZIP file is open in Archive View, you can compare it with another archive by using the

command [Tools | Compare Directories](#).

12.4 EPUB Files

An EPUB file is a zipped group of files conforming to the [EPUB standard](#) of the [International Digital Publishing Forum \(IDPF\)](#). This standard is the distribution and interchange standard for digital web publications. In [Archive View](#), you can open EPUB files, view the EPUB file's digital publication in a preview tab, edit component files of the EPUB archive directly in XMLSpy, validate the EPUB file, and save the component files back to the EPUB archive.

Note: (i) XMLSpy supports [EPUB 2.0.1](#). (ii) Sample EPUB files are available in the `Examples` project and in the (My) `Documents/Altova/XMLSpy2012/Examples` folder.

Terminology

In the descriptions below, terms are used as follows:

- **EPUB file** is used to indicate the EPUB file having the file extension `.epub`. This is the ZIP file that contains the whole archive and is the file that will be opened in Archive View
- An **archive file** is any one of the files contained in the EPUB archive
- **EPUB book** is the term used to indicate the digital publication generated by the zipped EPUB file

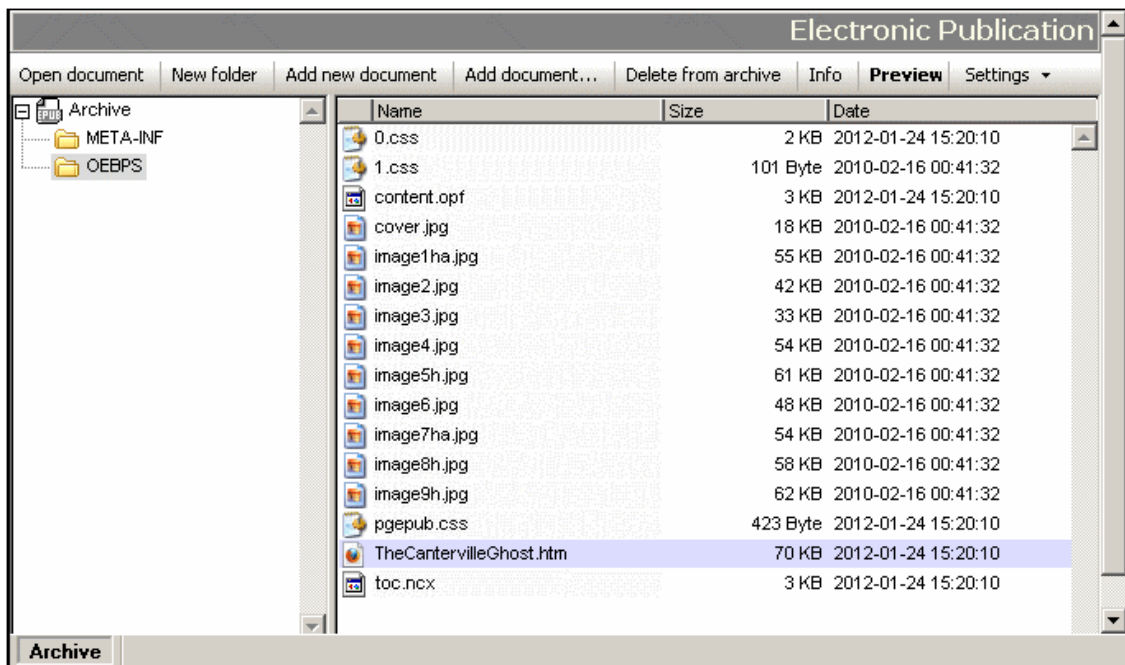
In this section

The description below of EPUB functionality in XMLSpy is structured into the following parts:

- [Opening EPUB files in Archive View](#)
- [Creating a new EPUB file](#)
- [Previewing an EPUB book](#)
- [Modifying the contents and structure of an EPUB archive](#)
- [Editing archive files directly in XMLSpy](#)
- [Entry helpers for archive files](#)
- [Validating EPUB file](#)

Opening EPUB files in Archive View

Select the menu command **File | Open**, navigate to the EPUB file, and click **Open**. The EPUB file opens in Archive View (*screenshot below*). Alternatively, you can right-click the EPUB file in Windows Explorer and select the context menu command to open the file with XMLSpy. If you have [set XMLSpy to be the default editor of EPUB files](#), then double-clicking the EPUB file will open the file in Archive View.



Folder View on the left-hand side shows the folders in the archive, whereas the Main Window shows the files in the folder selected in Folder View. The EPUB archive will have the following structure and the following key components.

```

Archive
|-- Mimetype file
|
|-- META-INF folder
|   |-- container.xml
|
|-- DOCUMENT folder (In the screenshot above, OEBPS is the Document folder.)
|   |-- Contains HTML, CSS, image files, plus OPF and NCX files

```

Creating a new EPUB file

To create a new EPUB file, select the menu command **File | New**. In the Create New Document dialog that pops up, select the file type `.epub`. In the Save As dialog that now pops up, give a name for your EPUB document and click **Save**. A skeleton EPUB archive containing all the folders and files of a valid EPUB archive (see *archive structure above*) will be created in a new window in Archive View. Add the folders and files you wish to add to the archive, as described below, and then save the EPUB file. To edit an archive file directly in XMLSpy, double-click the file in Archive View. The file will open in a new XMLSpy window. Edit it and then save it with the **File | Save (Ctrl+S)** command.

Previewing an EPUB book

To preview an EPUB book, make the EPUB file active in Archive View, then click the **Preview** button in the toolbar of Archive View. The EPUB book will open in an (Internet Explorer) browser window in XMLSpy. If any of the files that will be used for the preview—whether a content file or a structure-related file—has been modified but not yet saved, you will be prompted to save the file. If you do not save the modifications, the preview will use the previously saved data and might not be up-to-date. You can specify that all modified files be saved automatically before

previewing by toggling on this setting (via the **Settings** button in the the toolbar of Archive View).

Note the following:

- If the **Preview** button is clicked while a Preview window of that EPUB publication is still open, then the open Preview window will be updated and reloaded.
- Refreshing the Preview window will not update the Preview window. The **Preview** button of the corresponding EPUB file must be clicked to update the Preview window (*see previous point*).
- To close the preview, close the Preview window.

Note: Not all EPUB markup is supported in Internet Explorer, so previews could be distorted. Additionally, if the digital publication document is XML—and not HTML—the preview might not work. Newer versions of Internet Explorer provide improved handling of EPUB markup, so if you experience problems, try updating to the latest version of Internet Explorer.

Modifying the contents and structure of an EPUB archive

You can add folders (click the **New Folder** button), new files (**Add New Document**), and existing files (**Add Document**) to the selected archive folder. Note that when you add a new folder or new document, you must immediately enter a name for the folder or file; it is not possible to rename folders in Archive View. You can delete a file or folder by selecting it and clicking the **Delete from Archive** button.

After you have modified the archive you must save the EPUB file (**File | Save**) for the changes to be saved.

Editing archive files directly in XMLSpy

To edit an archive file directly in XMLSpy, double-click the file in Archive View. Alternatively, select the file in Archive View and click the Open Document button in the toolbar of Archive View. The file will open in a new XMLSpy window. Edit it and then save it with the **File | Save (Ctrl+S)** command.

Entry helpers for archive files

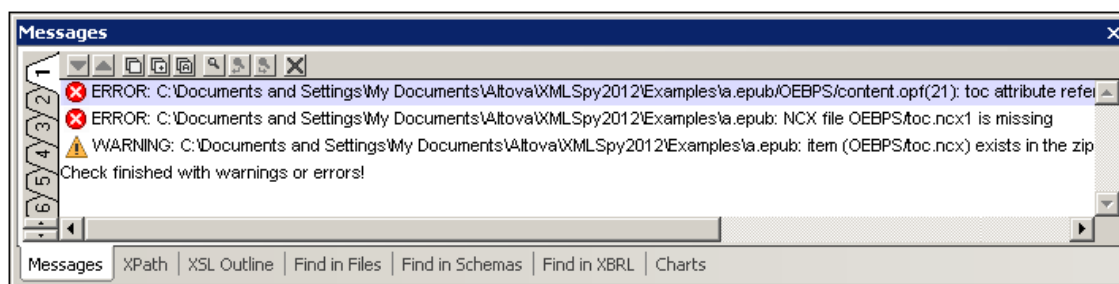
Entry helpers for standards-based archive files are available when these archive files are opened in XMLSpy. These archive files are:

- The OPF file, traditionally named `content.opf`, contains the EPUB book's metadata. It is based on the [Open Packaging Format \(OPF\) specification](#).
- The NCX file (Navigation Control file for XML), traditionally named `toc.ncx`, contains the publication's table of contents. It is based on the [NCX part](#) of the OPF specification.
- The folder named `META-INF` must contain the file `container.xml`, which points to the file defining the contents of the book (the OPF file). The file `container.xml` specifies how the archive files should be organized according to rules in the [Open Container Format \(OCF\) specification](#).

Validating an EPUB file

To validate an EPUB file, select the command **XML | validate XML (F8)**. The validation results

are displayed in the Messages window (*screenshot below*). If any of the archive files—whether a content file or a structure-related file—has been modified but not yet saved, you will be prompted to save the file. You must save the modified files in order to validate the EPUB file. You can specify that all modified files be saved automatically before validation by toggling on this setting (via the **Settings** button in the the toolbar of Archive View).



Error messages display: (i) the file in which the error was found, and, if applicable, the number of the line in which the error occurs; (ii) a description of the error. In the screenshot above, the highlighted error occurs in line 21 of the file `content.opf`. Clicking on the error line in the Messages window opens the relevant file and highlights the error.

Note: The EPUB validation engine is a Java utility, so Java must be installed on your machine for the validation engine to run.

13 Databases

XMLSpy enables you to connect to a variety of databases (DBs) and then perform operations such as querying the DB, importing the DB structure as an XML Schema, generating an XML data file from the DB, and exporting data to a DB. Each DB-related feature is available in XMLSpy as a menu command, and is described in the [User Reference](#) section of this documentation under the respective command. A complete list of these command is given below, with links to the respective descriptions.

In this section, we do the following:

- Describe [how to connect to a database](#), which is an operation that is required for executing any of XMLSpy's DB-related commands; and
- [List DBs](#) that have been successfully tested for use with XMLSpy.

Note: If you are using the 64-bit version of XMLSpy, ensure that you have access to the 64-bit database drivers needed for the specific database you are connecting to.

XMLSpy's DB-related features

XMLSpy's DB-related features are executed with commands in the [DB](#) and [Convert](#) menus.

- [Query Database](#): In the **DB** menu. Loads the structure of the DB in a separate Database Query window and enables queries to the DB. Results are displayed in the Database Query window.
- [IBM DB2](#): In the **DB** menu. IBM DB2 is an XML DB, and XMLSpy enables management of the XML Schemas of the XML DB as well as editing and validation of the XML DB.
- [Oracle XML DB](#): In the **DB** menu. Provides a range of functionality for Oracle XML DBs, including XML Schema management, database querying, and generation of XML files based on DB schemas.
- [Import Database Data](#): In the **Convert** menu. Imports DB data into an XML file.
- [Create XML Schema from DB Structure](#): In the **Convert** menu. Generates an XML Schema that is based on the structure of the DB.
- [DB Import Based on XML Schema](#): In the **Convert** menu. With an XML Schema document active in XMLSpy, a DB connection is made and the data of a selected DB table can be imported. The resulting XML document will have a structure based on the XML Schema that was active when the DB connection was made.
- [Create DB Structure from XML Schema](#): In the **Convert** menu. DB tables with no data are created based on the structure of an existing XML Schema.
- [Export to Database](#): In the **Convert** menu. Data from an XML document can be exported to a DB. Existing DB tables can be updated with the XML data, or new tables can be created that contain the XML data.

Datatype conversions

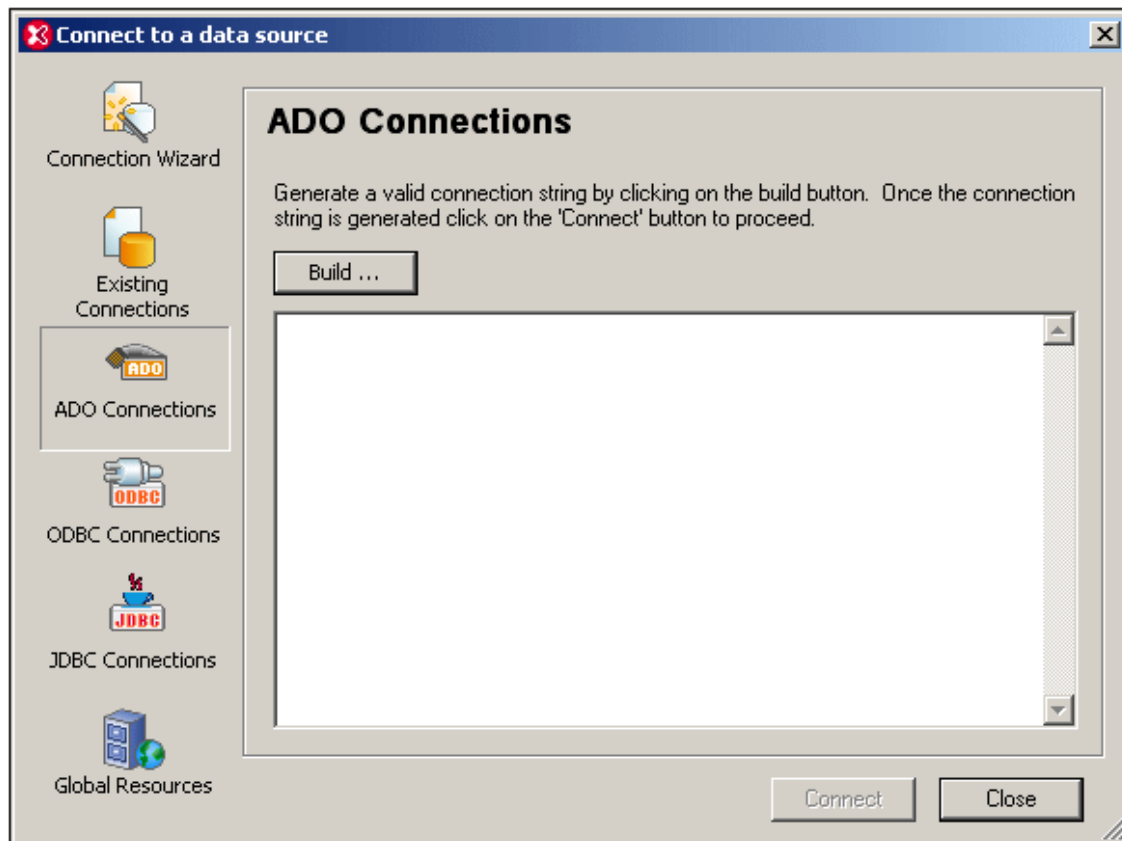
When converting data between XML documents and DBs, datatypes must necessarily be converted to types appropriate for the respective formats. The way XMLSpy converts datatypes is given in the appendices [Datatypes in DB-Generated XML Schemas](#) and [Datatypes in DBs Generated from XML Schemas](#).

Altova DatabaseSpy

Altova's DatabaseSpy is a multi-database query and DB design tool that offers additional DB functionality to that available in XMLSpy. For more details about Altova DatabaseSpy, visit the [Altova website](#).

13.1 Connecting to a Data Source

A number of commands and icons in the **DB** and **Convert** menus require a connection to a database. XMLSpy enables you to connect to a variety of databases (*see list below*). It uses the Connect to Data Source dialog (for commands in the **Convert** menu, *screenshot below*) or Quick Connect dialog (for the **DB | Query Database** command) to set up and make the connection. Both dialogs are essentially the same; they guide you through the same connection steps. In this section, we describe how the mechanism behind these two dialogs works. In descriptions of the **DB** or **Convert** commands that require a database connection, refer to this section for information on making the connection.



The options for connecting to the database are:

- Using a [Connection Wizard](#) that guides you through the connection process
- Using an [existing connection](#)
- Using an [ADO Connection](#)
- Using an [ODBC Connection](#)
- Using a [JDBC Connection](#)
- Using a [global resource](#)

Each option is described in the subsections of this section. The descriptions in this section explain only the connection mechanism. The dialogs that appear subsequent to the connection process are dependent on the specific command being executed, and these dialogs are therefore described in the sections relating to that specific command. For example, if you are looking for a description of the command [Convert | Import Database Data](#), the description for the connection process is in this section, but the selection of DB tables is described in the section [Convert | Import Database Data](#).

Note: If you are using the 64-bit version of XMLSpy, ensure that you have access to the 64-bit database drivers needed for the specific database you are connecting to.

13.1.1 Connection Wizard

In the Connect to Data Source dialog, when the **Connection Wizard** button is selected, the Connection Wizard (*screenshot below*) pops up. The Connection Wizard helps you to connect to the most commonly used types of databases. In this section, we go through the connection to a Microsoft Access database and an IBM DB2 database.

MS Access

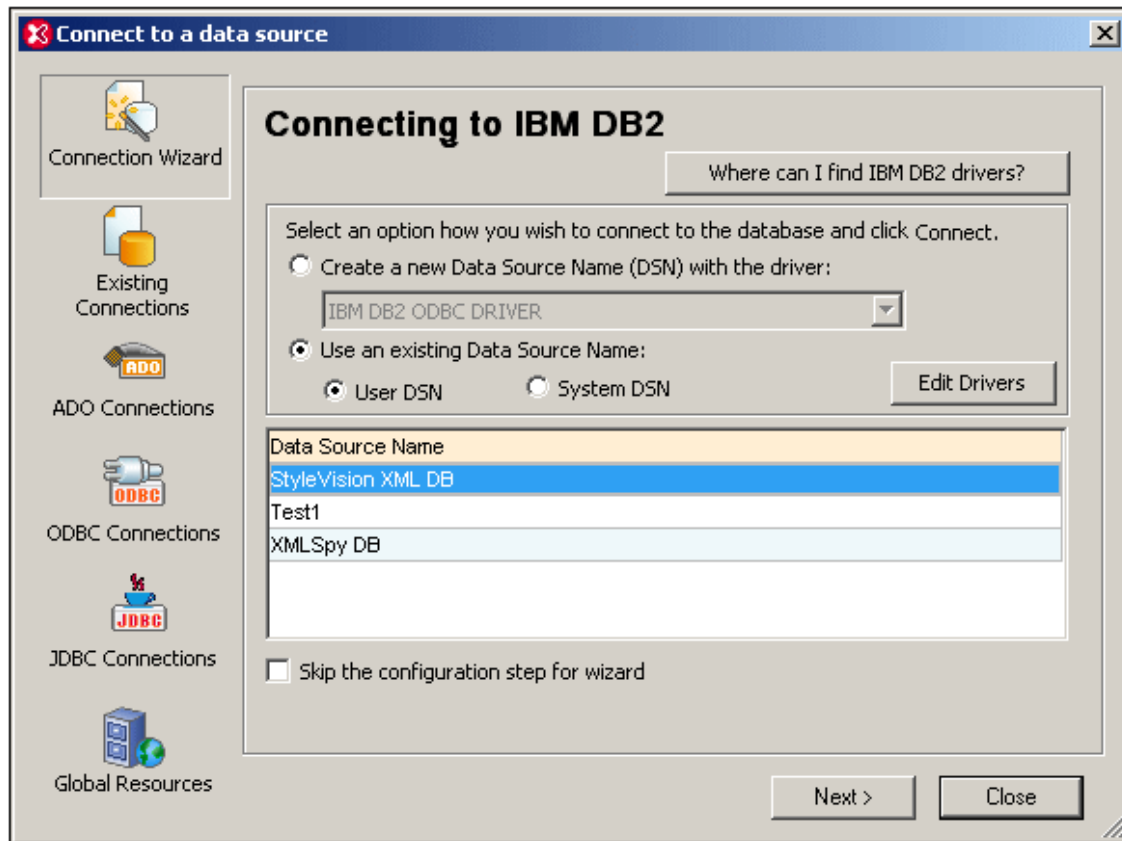
Carry out the following steps to connect to an MS Access database. Select Microsoft Access (ADO) (*screenshot below*), and click **Next**.



In the Connect to MS Access dialog that pops up, browse for the MS Access database, and click **Next**. The connection to the data source is established. For information about subsequent dialogs, refer to the description of the command being executed.

IBM DB2

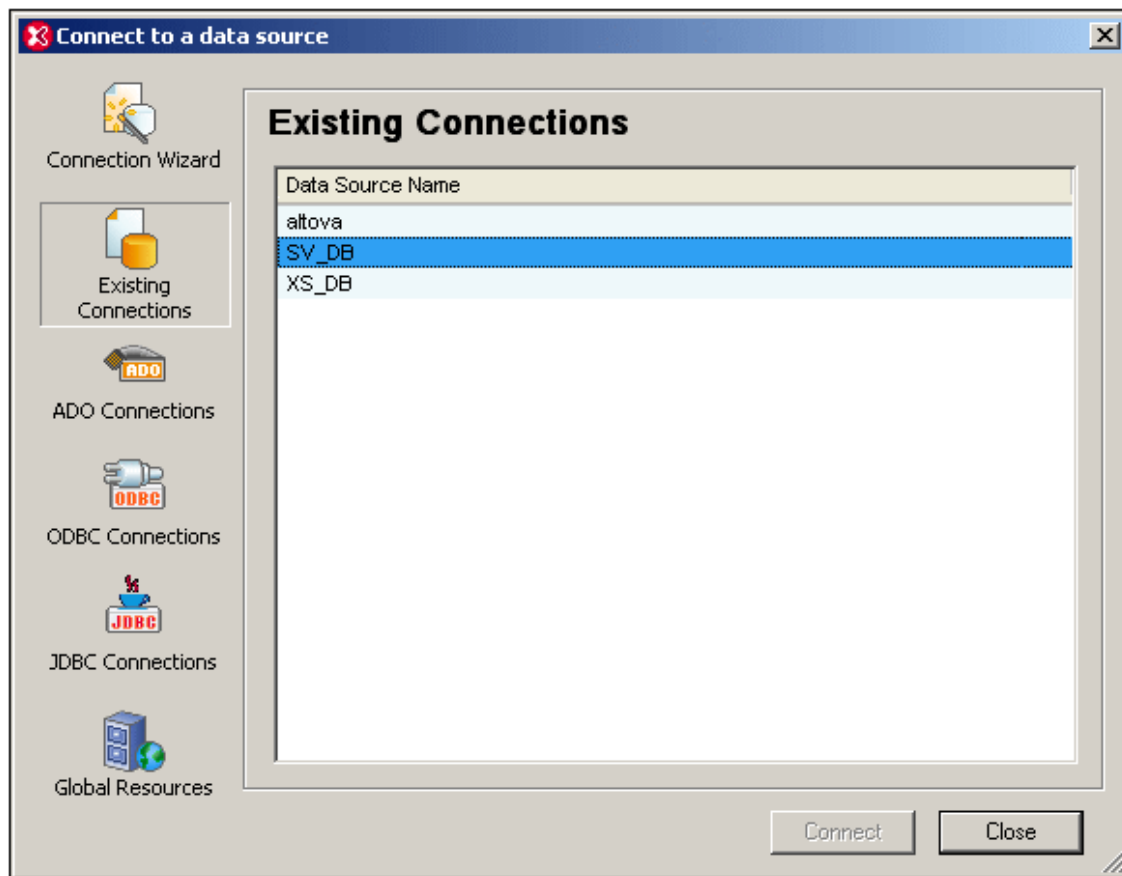
In the first screen of the Connection Wizard (*see first screenshot in this section*), select IBM DB2 and click **Next**. The Configuration dialog (*screenshot below*) pops up. The wizard will then guide you in configuring the connection to the IBM DB2 database and making the connection. These steps consist essentially of selecting the appropriate driver to connect to the DB, then locating the DB, and entering user information that enables you to connect.



In the Configuration dialog above, select an existing data source, or create a new data source with an appropriate driver (additional drivers can be added to the list of available drivers by clicking the **Edit Drivers** button and selecting the required drivers). In the following screen you will be prompted for user information (ID and password). Clicking **OK** will establish the connection with the database. Also see [ODBC Connections](#) for more details. For information about subsequent dialogs, refer to the description of the command being executed.

13.1.2 Existing Connections

In the Connect to Data Source dialog, when the **Existing Connections** button is selected, the Existing Connections pane opens (*screenshot below*), showing all the connections that are currently established.

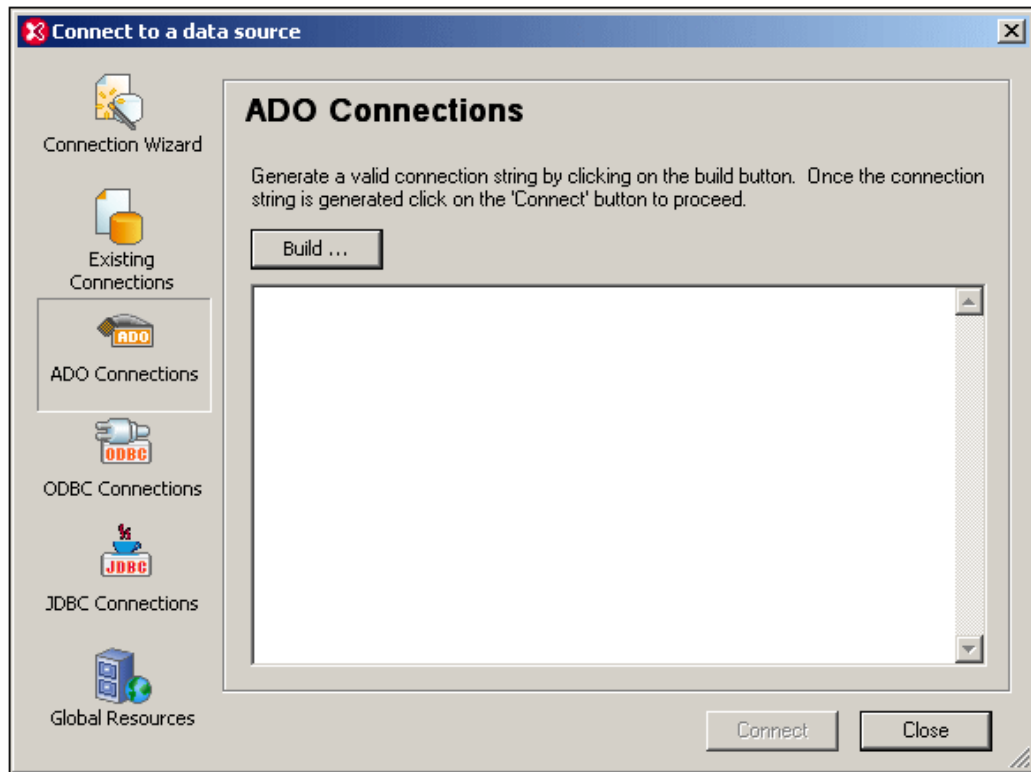


Select the required database and click **Connect**. The connection to the database is established. For information about subsequent dialogs, refer to the description of the command being executed.

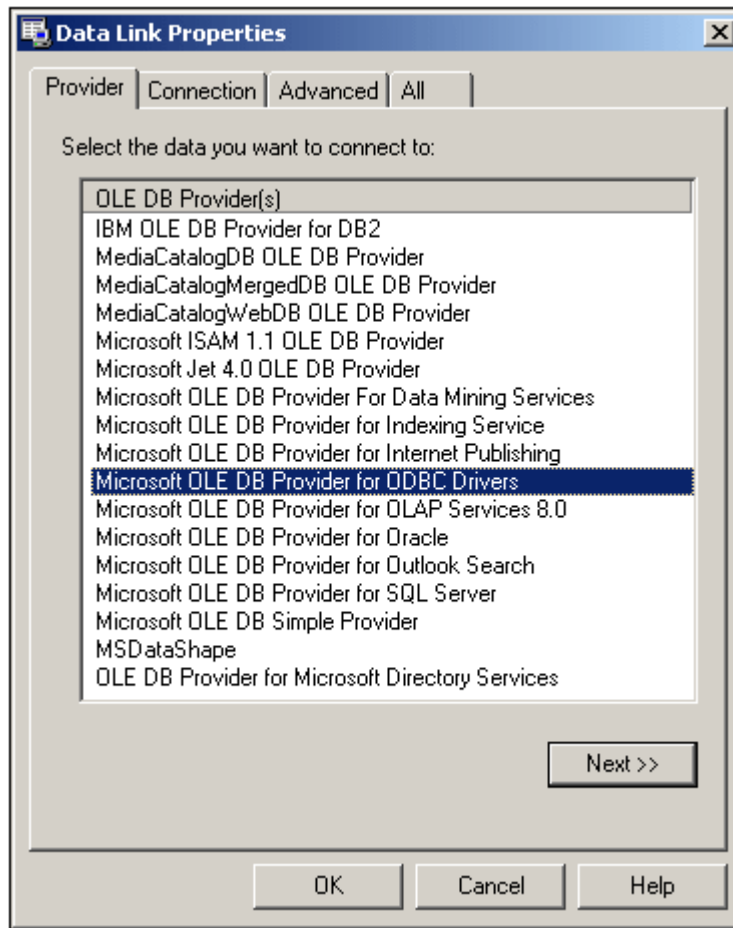
13.1.3 ADO Connections

The ADO Connections option enables you to connect to a DB via ADO. In this section, the connection via ADO is described using an IBM DB2 database as the target DB. Where options are different if another type of database is the target DB, these differences are noted.

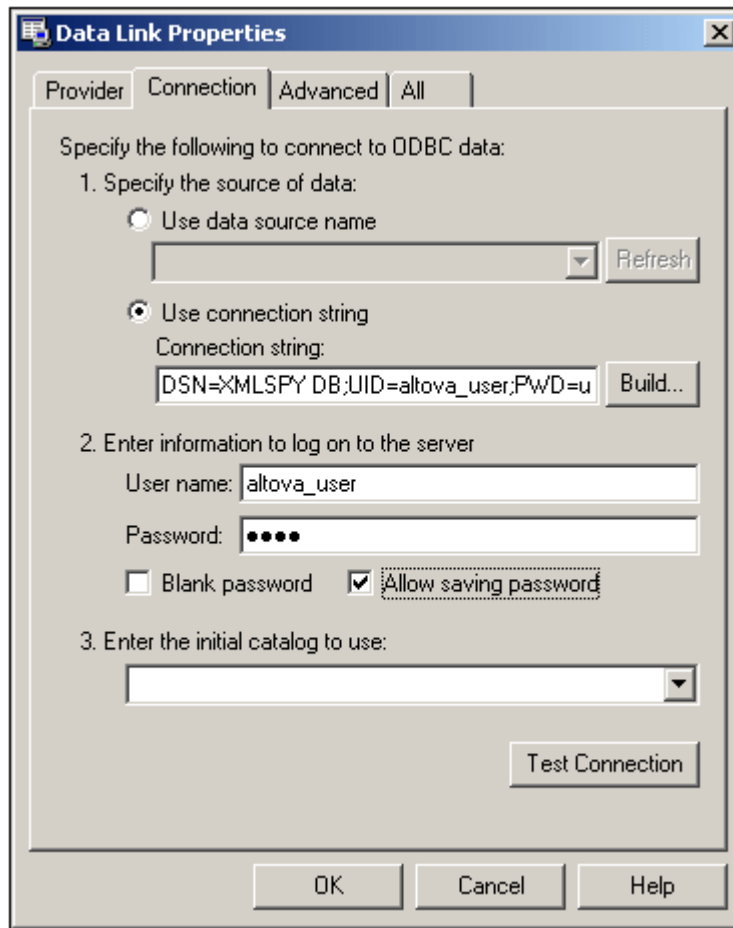
1. In the Connect to Data Source dialog, select ADO Connections. The ADO Connections box appears (*screenshot below*).



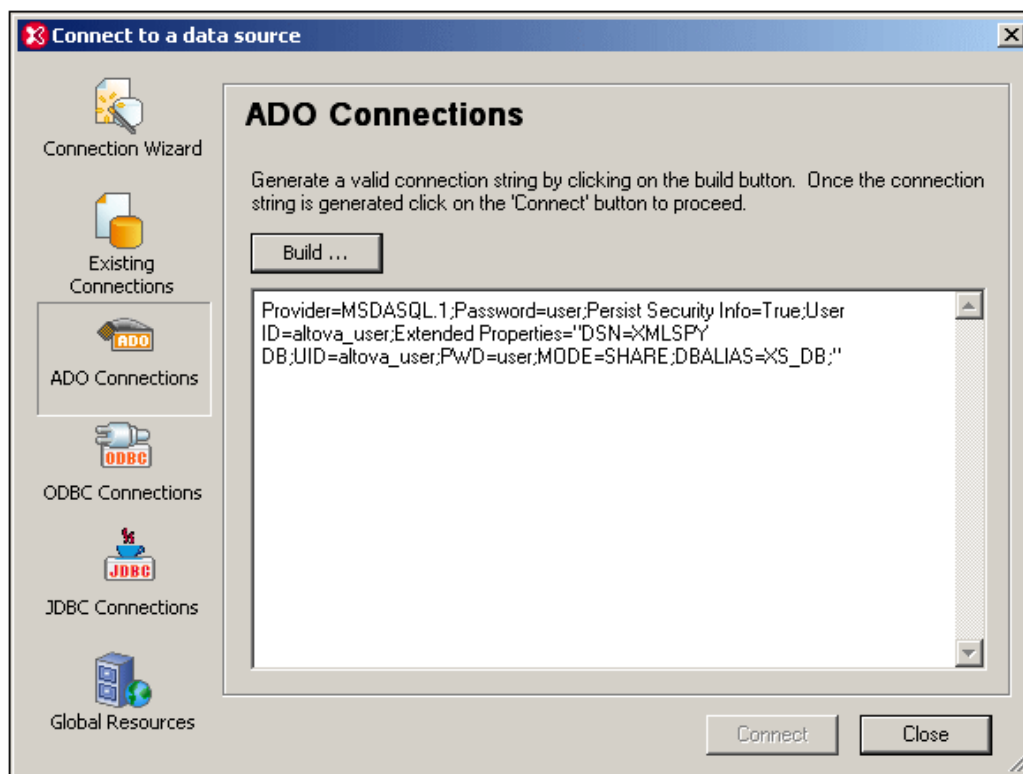
2. Click the **Build** button. The Data Link Properties dialog (*screenshot below*) opens.



3. Select Microsoft OLE DB Provider for ODBC Drivers from the list and click **Next**. The Connection tab is activated. (For Microsoft SQL DBs, we recommend *Microsoft OLE DB Provider for SQL Server*; for Oracle, MySQL, Sybase, and IBM DB2 DBs, *Microsoft OLE DB Provider for ODBC Drivers*.)



4. Build a connection string via the **Build** button (the dialog which pops up prompts you for a data source and user information). In the case of some databases (such as Microsoft SQL Server), you do not build a connection string but instead select the server and database from combo box listings.
5. If login information is required, enter a user name and password, and check the Allow Saving Password check box.
6. Click **OK**. The Data Link Properties dialog closes. The connection string appears in the ADO Connections box (*screenshot below*).



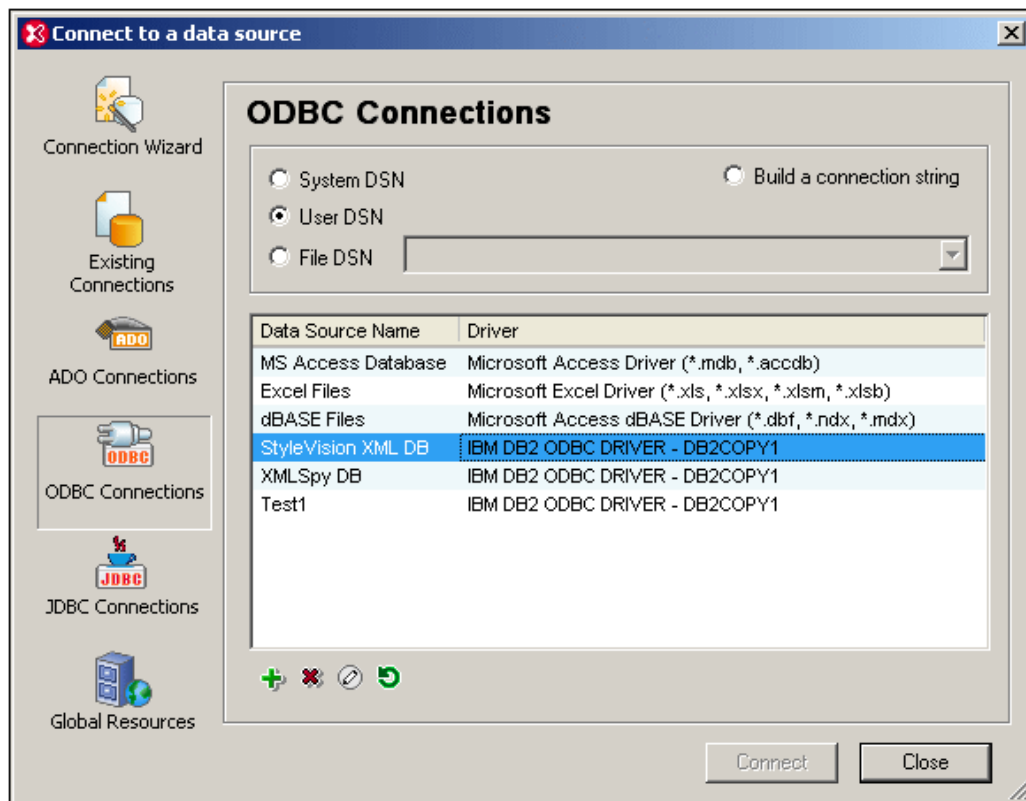
7. Click the **Connect** button. The connection to the data source is established. For information about subsequent dialogs, refer to the description of the command being executed.

Note: For an ADO connection to MS SQL Server via the SQL Server Native Client 10.0 driver the following property values must be set in the *All* tab of the Data Link Properties dialog: (i) Set the property value of Integrated Security to a space character; (ii) Set the property value of Persist Security Info to `true`.

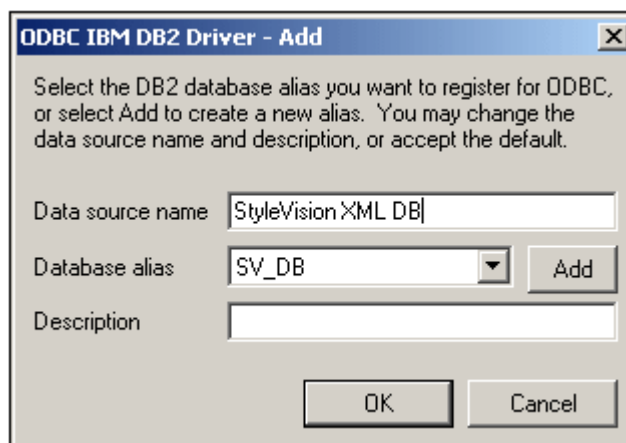
13.1.4 ODBC Connections

This section describes how to connect to a DB via ODBC. The steps listed below describe a connection to an IBM DB2 database, but apply also to other types of databases that can be connected via ODBC. To connect using an ODBC connection, do the following:

1. In the Connect to Data Source dialog, select **ODBC Connections**.



2. Select one of the options for specifying a Data Source Name (DSN). If you select System DSN or User DSN, the available DSNs are displayed in the Data Source pane. If you select File DSN, you are prompted to browse for the DSN file. Alternatively, you can build a connection string to the DB by selecting the Build a Connection String option.
3. If you wish to add a DSN to those in the Data Source pane, click the Create a New DSN icon .
4. In the Create an ODBC DSN dialog that appears, select the required driver, then click the **User DSN** or **System DSN** button.
5. In the dialog that appears (*screenshot below*), select the DB alias and give it a DSN.



6. Click **OK** to finish. The DB is added as a Data Source to the list in the Data Source pane.
7. After you have selected the DataSource (via the System DSN or User DSN option), or

selected a DSN File (File DSN option), or built a connection string (Connection String option), click **Connect**.

8. When you are prompted for your user ID and password, enter these and then click **OK**. The connection is established. For information about subsequent dialogs, refer to the description of the command being executed.

Note: The listing in the data source pane (when the System DSN or User DSN option is selected) can be edited using the *Edit Selected DSN*, *Remove Selected DSN*, and *Refresh Listed DSNs* icons at the bottom of the ODBC Connections screen.



Edit selected DSN.



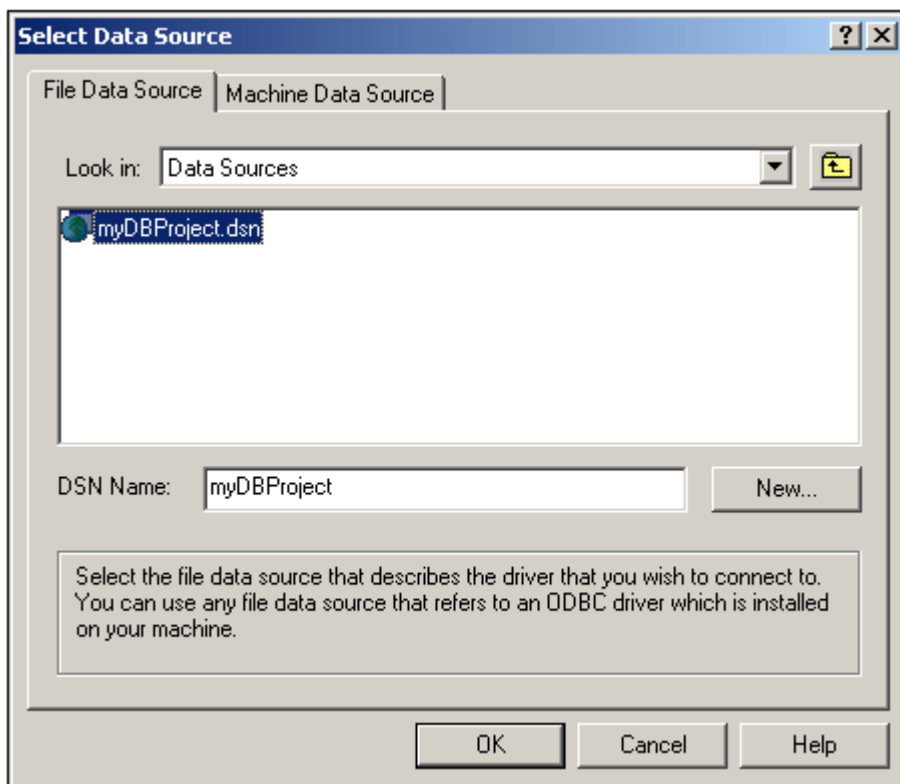
Remove selected DSN.



Refresh listed DSNs.

Building a connection string

In the ODBC Connections screen, when you select the Build a Connection String radio button and click the **Build** button that appears, the Select Data Source dialog (*screenshot below*) pops up. You can either select a File DSN (in the File Data Source tab), or select a data source that is available on your machine (listed in the Machine Data Source tab).



Clicking **OK** pops up a dialog that prompts for user information, including the User ID and password. When you click **OK** in this dialog, the connection string is entered in the ODBC Connections screen.

13.1.5 JDBC Connections

This section describes how to connect to a DB via JDBC. The steps listed below describe a connection to an IBM DB2 database, but they apply also to other types of databases that can be connected to via JDBC.

Pre-connection steps

Before connecting to the DB via JDBC, you must do the following:

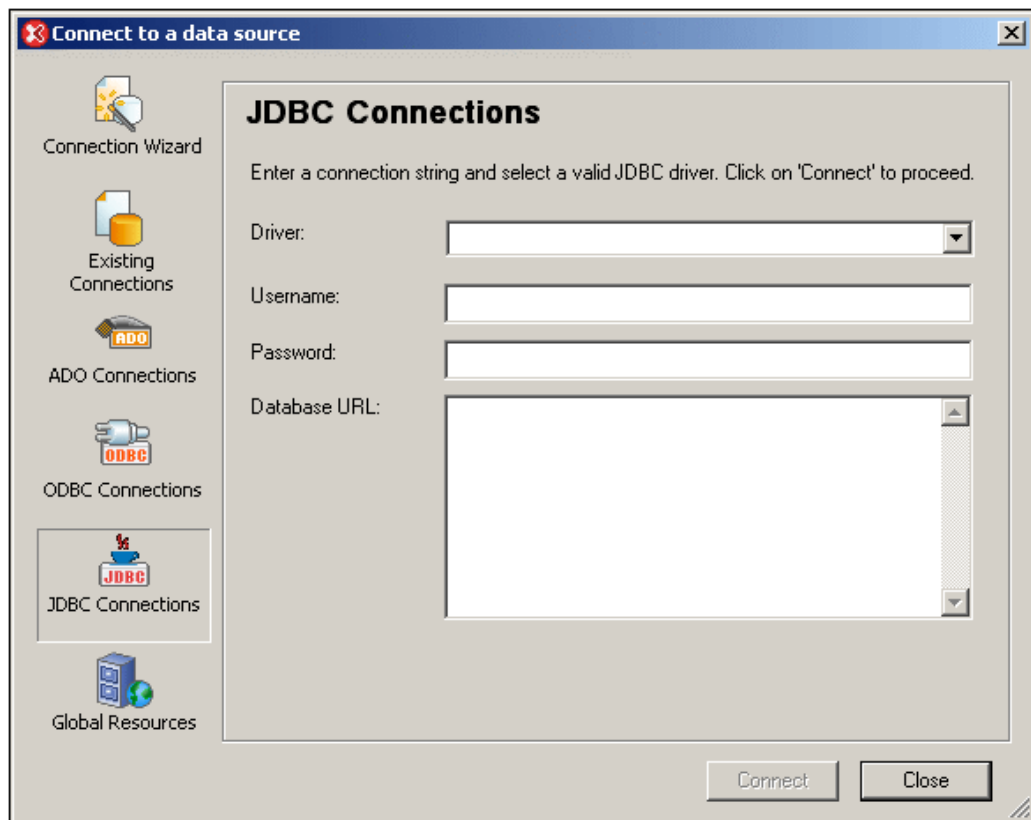
1. Ensure that the Java Runtime Environment (JRE) is installed. If it isn't installed, then install it. Use a 32-bit JRE for a 32-bit machine or a 64-bit JRE for a 64-bit machine.
2. Install a JDBC driver. No special installation setup is required. All you need to do is copy the driver to a local directory, for example, `c:\jdbc`. Note that JDBC drivers (which are Jar files) are platform-independent.
3. Set up the `CLASSPATH` to include the location where the JDBC driver is located. (The application reads the `CLASSPATH` environment variable to locate the JDBC driver.) To access and edit the `CLASSPATH`, do the following: Click **Start | Control Panel | System | Advanced | Environment Variables**. In the Environment Variables dialog that appears, select either the `CLASSPATH` user environment variable or the `CLASSPATH` system environment variable and click the **Edit** button. Add the path to the JDBC driver to the `CLASSPATH`. For example: `CLASSPATH=C:\jdbc\sqljdbc.jar;C:\jdbc\db2jcc.jar;`
4. Log off and then log on again to make the `CLASSPATH` changes active.

Note: Installing the complete IBM DB2 or Oracle client will automatically populate the `CLASSPATH` with the JDBC drivers of the respective packages. For more details, see the summaries below.

Connecting via JDBC

To connect using a JDBC connection, do the following:

1. In the Connect to a Data Source dialog, select **JDBC Connections**. The JDBC Connections pane (*screenshot below*) appears.



2. Select a JDBC driver from the *Driver* dropdown list (all detected drivers, i.e. those in the `CLASSPATH` environment variable, are listed). Enter a connection string in the *Database URL* text box and a user name and password as required. Given below is the connection string syntax for commonly used databases, each with an example connection string.

Oracle `jdbc:oracle:thin:[ user/password]@//[ host][: port]/SID`
 `jdbc:oracle:thin:@//abcd234/ORA11`

IBM DB2 `jdbc:db2://host_name: port/dbname`
 `jdbc:db2://MyDB2: 50000/boz`

MySQL `jdbc:mysql://host_name: port/dbname`
 `jdbc:mysql://MyDB2: 3306/moz`

MSSQL `jdbc:sqlserver://host: port; databasename=name; user=name; pass`
 `word=Pwd`
 `jdbc:sqlserver://abcd38: 1433; databasename=coz`
 `jdbc:sqlserver://Q5; DatabaseName=coz; SelectMethod=Cursor`

PostgreSQL jdbc:postgresql://host:port/database
L

jdbc:postgresql://abc993:5432/qanoz

Sybase jdbc:sybase:Tds:host:port/dbname

jdbc:sybase:Tds:abc12:2048/QUE

3. Click the **Connect** button. The connection to the data source is established. For information about subsequent dialogs, refer to the description of the command being executed.

Step-by-step: MSSQL, MySQL, PostGre, and other non-XML DBs

Given below is a step-by-step guide for connecting to a non-XML DB via JDBC:

1. JDBC JAR files (driver files): Copy the files to any local location. Then add the full path and filename to the Windows `CLASSPATH` variable. For example:
C:\jdbc\sqljdbc.jar; C:\jdbc\db2jcc.jar;
2. Log off and log on to make the `CLASSPATH` changes active.
3. Start XMLSpy and access the Connect to a Data Source dialog.
4. The JDBC Connections pane lists, in the *Driver* dropdown box, the detected JDBC drivers. If the dropdown box is empty, make sure that the file `altovadb.jar` is present in the folder: C:\Program Files\Altova\Common2012\jar.
5. Connect to a database as described in the section *Connecting via JDBC* above.

Step-by-step: Oracle

Given below is a step-by-step guide for connecting to an Oracle DB via JDBC. If you don't need the XML and XDB features of the Oracle DB, follow the instructions in the step-by-step guide for non-XML DBs. The installation folder of the Oracle client is indicated in the steps below by the placeholder: `%ORACLE_HOME%`.

1. Install Oracle client software with OCI and ODBC features enabled. If an Oracle client is already installed check if the two jar files below are present:
`%ORACLE_HOME%\LIB\xmlparserv2.jar`
`%ORACLE_HOME%\RDBMS\jlib\xdb.jar`
2. Add the following files to the Windows `CLASSPATH` environment variable:
`%ORACLE_HOME%\jdbc\lib\ojdbc6.jar`
`%ORACLE_HOME%\LIB\xmlparserv2.jar`
`%ORACLE_HOME%\RDBMS\jlib\xdb.jar`
3. Log off and log on to make the `CLASSPATH` changes active.
4. Start XMLSpy and access the Connect to a Data Source dialog.
5. The JDBC Connections pane lists, in the *Driver* dropdown box, the detected JDBC drivers. If the dropdown box is empty, make sure that the file `altovadb.jar` is present in the folder: C:\Program Files\Altova\Common2012\jar.
6. Connect to a database as described in the section *Connecting via JDBC* above.

Step-by-step: IBM DB2

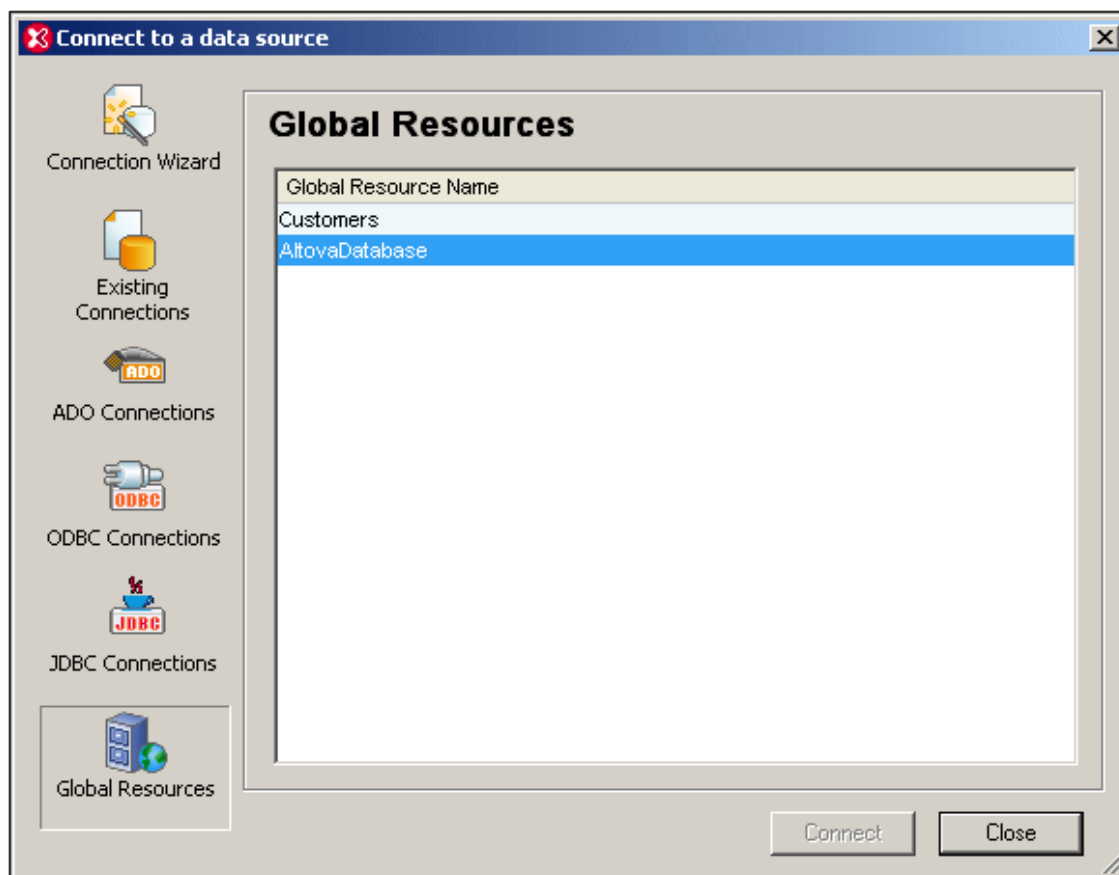
Given below is a step-by-step guide for connecting to an IBM DB2 database via JDBC.

1. If an IBM DB2 client has already been installed, nothing needs to be done since the `CLASSPATH` will have been set by the installation process.
2. If no IBM DB2 client has been installed, add the IBM DB2 JDBC driver jar files `db2jcc.jar` and `db2jcc_license_cu.jar` to the Windows `CLASSPATH`. Log off and log on to make the `CLASSPATH` changes active.
3. Start XMLSpy and access the Connect to a Data Source dialog.
4. The JDBC Connections pane lists, in the *Driver* dropdown box, the detected JDBC drivers. If the dropdown box is empty, make sure that the file `altovadb.jar` is present in the folder: `C:\Program Files\Altova\Common2012\jar`.
5. Connect to a database as described in the section *Connecting via JDBC* above.

Note: When databases are connected to via JDBC, due to insufficient information returned by the drivers: (i) data editing is not possible for tables without a primary key; (ii) Execute for data-editing in SQL Editor will not work.

13.1.6 Global Resources

In the Connect to Data Source dialog, when the **Global Resources** button is selected, the Global Resources pane opens (*screenshot below*), showing all the database-type global resources that are defined in the currently active [Global Resources XML File](#).



Select the required global resource and click **Connect**. The connection to the database is established. For information about subsequent dialogs, refer to the description of the command

being executed.

13.2 Supported Databases

Altova XMLSpy fully supports the databases listed below. While Altova endeavors to support other ODBC/ADO databases, successful connection and data processing have only been tested with the listed databases.

The available root object for each of the supported databases is also listed.

Database (natively supported)	Root Object
MS SQL Server 2000, 2005, and 2008	database
MS SQL Server 2005 and 2008	schema
Oracle 9i, 10g, and 11g	schema
MS Access 2003 and 2007	database
MySQL 4.x and 5.x	database
IBM DB2 8.x and 9	schema
Sybase 12	database
IBM DB2 for i 5.4 and i 6.1	schema
PostgreSQL 8.0, 8.1, 8.2, 8.3	database

Note: If you are using the 64-bit version of XMLSpy, ensure that you have access to the 64-bit database drivers needed for the specific database you are connecting to.

14 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. For example, an XSLT stylesheet in XMLSpy can be used to transform an XML file generated on-the-fly by an Altova MapForce mapping.

Using Altova Global Resources involves two processes:

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

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.

14.1 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 **Tools | Customize** to display the Customize dialog. Then select the **Toolbars** tab and check the Global Resources option. This switches on the display of the Global Resources toolbar (*screenshot below*).



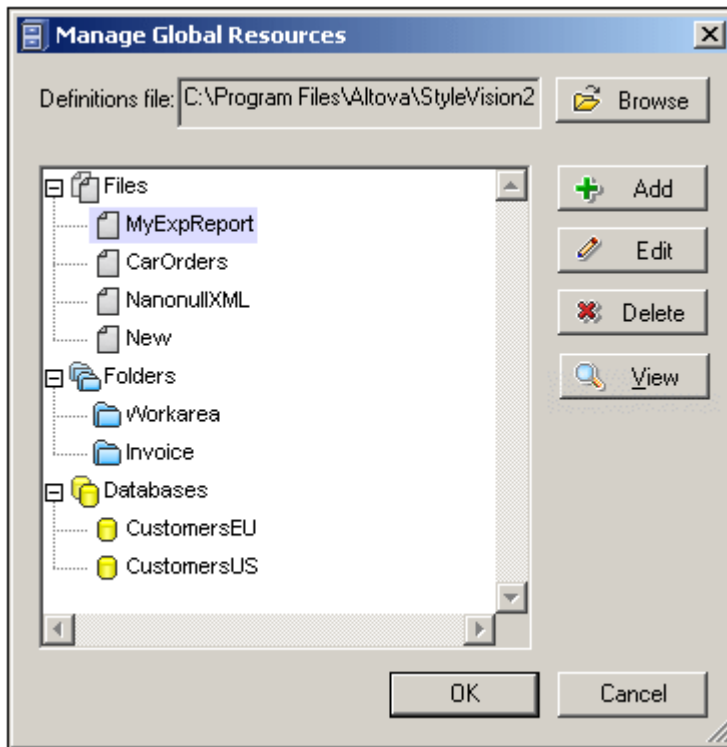
Once the toolbar is displayed, click the Manage Global Resources icon. This pops up the Global Resources dialog.

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.



Managing global resources: adding, editing, deleting

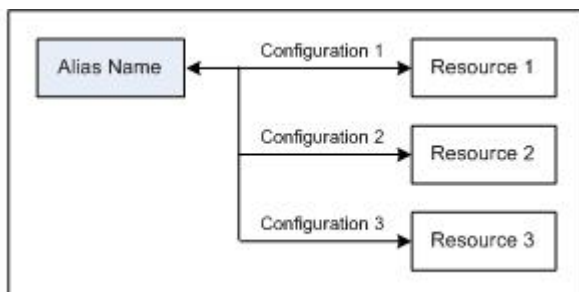
In the Manage Global Resources dialog (*screenshot above*), you can add a global resource to the selected Global Resources XML File, or edit or delete a selected global resource. The Global Resources XML File organizes the aliases you add into a list of several sections: files, folders, and databases (*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

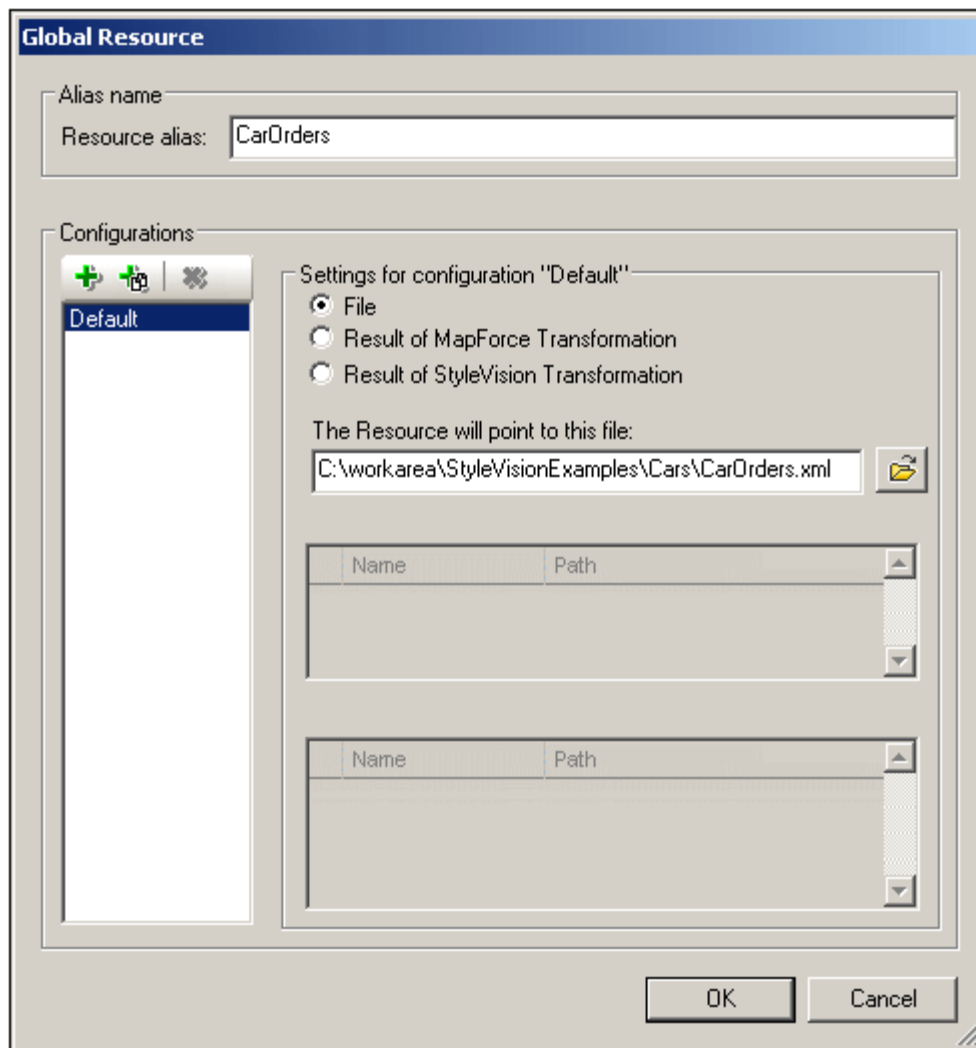
Creating a global resource involves mapping one alias name to one or more resources (file, folder, or database). Each mapping is called a configuration. A single alias name can therefore be associated with several resources via different configurations (*screenshot below*).



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.

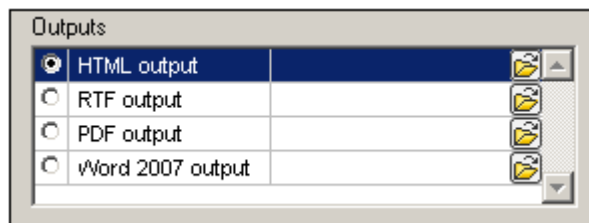
14.1.1 Files

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 StyleVision is installed (*the screenshot below shows the outputs for Enterprise Edition*).



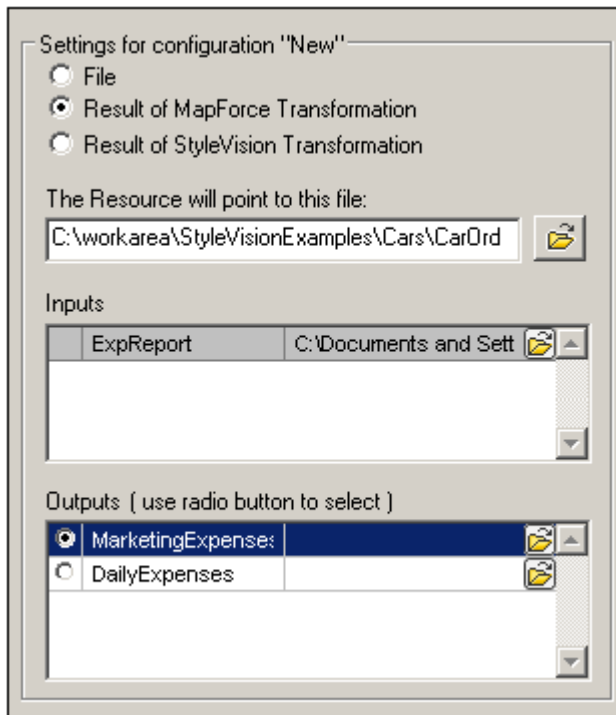
Select the radio button of the desired option (in the screenshot above, 'HTML output' is selected). The result of a transformation can itself be saved as a global resource or as a file path (click the  icon and select, respectively, Global Resource or Browse). If neither of these two saving options is 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.

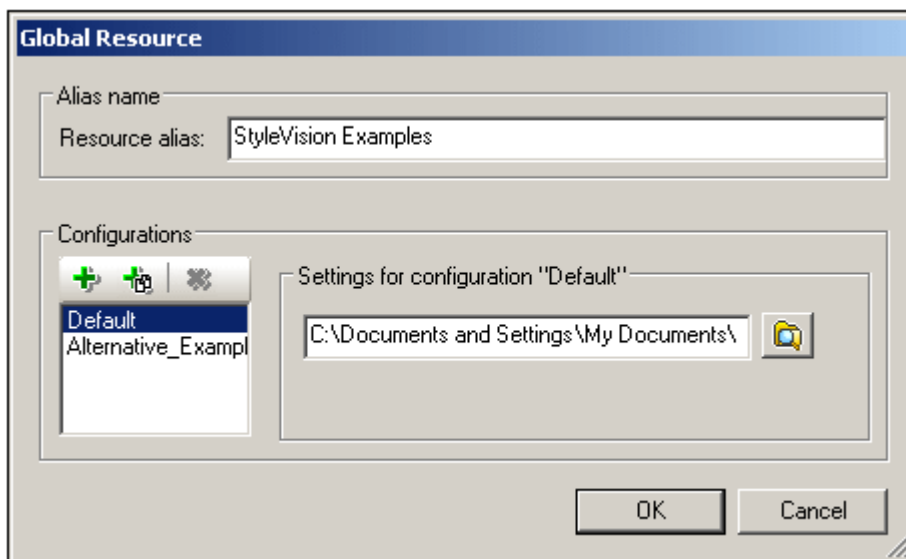
In a MapForce transformation that has multiple output schemas, you can select which one of the output schemas should be used for the global resource by clicking its radio button (*screenshot below*). The XML file that is generated for this schema can be saved as a global resource or as a file path (click 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).

14.1.2 Folders

In the Global Resource dialog for Folders (*screenshot below*), you can add a folder resource as follows:



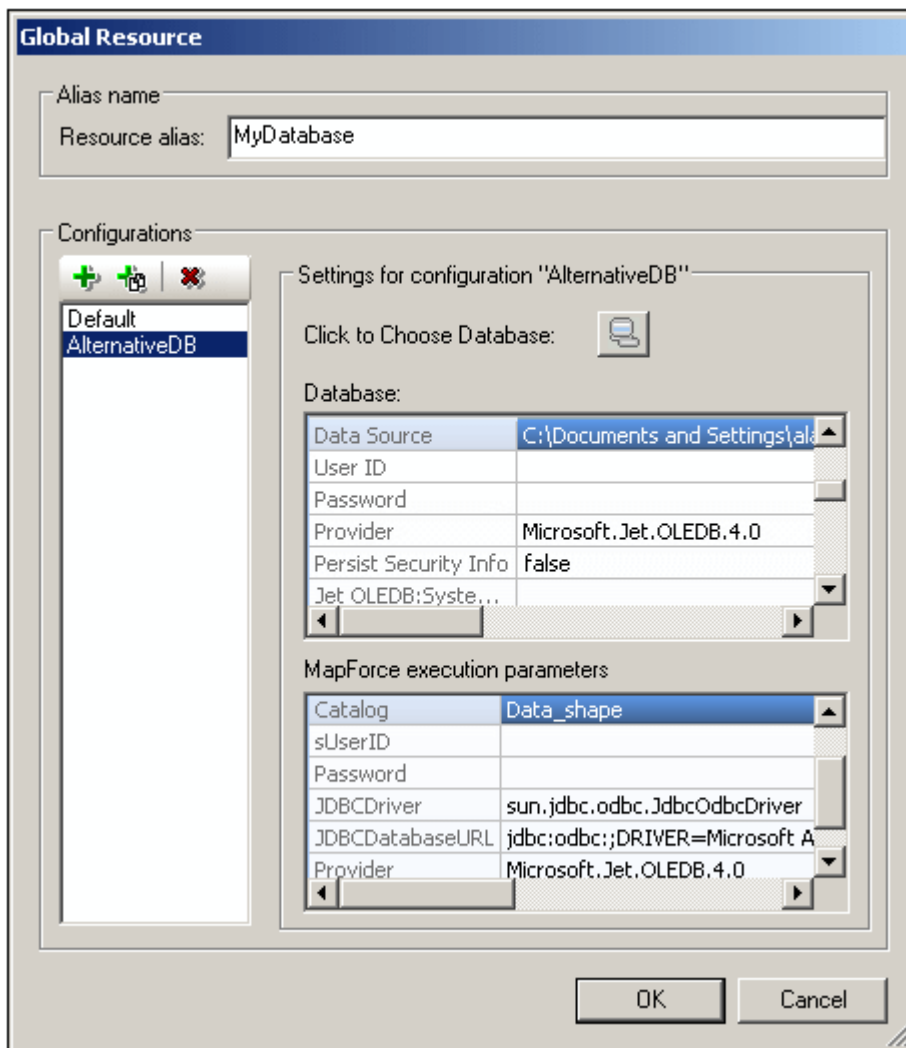
Enter an alias name.

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.

14.1.3 Databases

In the Global Resource dialog for Databases (*screenshot below*), you can add a database resource as follows:



To define a database-type global resource:

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).

3. Select one of the configurations in the Configurations pane and click the **Choose Database** icon. This pops up the Create Global Resources Connection dialog (*screenshot below*).



4. Select whether you wish to create a connection to the database using the Connection Wizard, an existing connection, an ADO Connection, or an ODBC Connection. Complete the definition of the connection method as described in the section [Connecting to a Database](#). You can use either the [Connection Wizard](#), [ADO Connections](#), or [ODBC Connections](#). If a connection has already been made to a database from XMLSpy, you can click the [Existing Connections icon](#) and select the DB from the list of connections that is displayed.
5. If you connect to a database server, you will be prompted to select a DB Root Object on the server. The Root Object you select will be the Root Object that is loaded when this configuration is used. If you choose not to select a Root Object, then you can select the Root Object at the time the global resource is loaded.
6. Specify a database resource for each configuration (that is, repeat Steps 3 and 4 above for the various configurations you have created).
7. Click **OK** in the **Global Resource** dialog to save the alias and all its configurations as a global resource. The global resource will be listed under databases in the Manage Global resources dialog.

14.1.4 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.

14.2 Using Global Resources

There are several types of global resources (file-type, folder-type , and database-type). Particular scenarios in XMLSpy 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 XMLSpy are listed in this section: [Files and Folders](#) and [Databases](#).

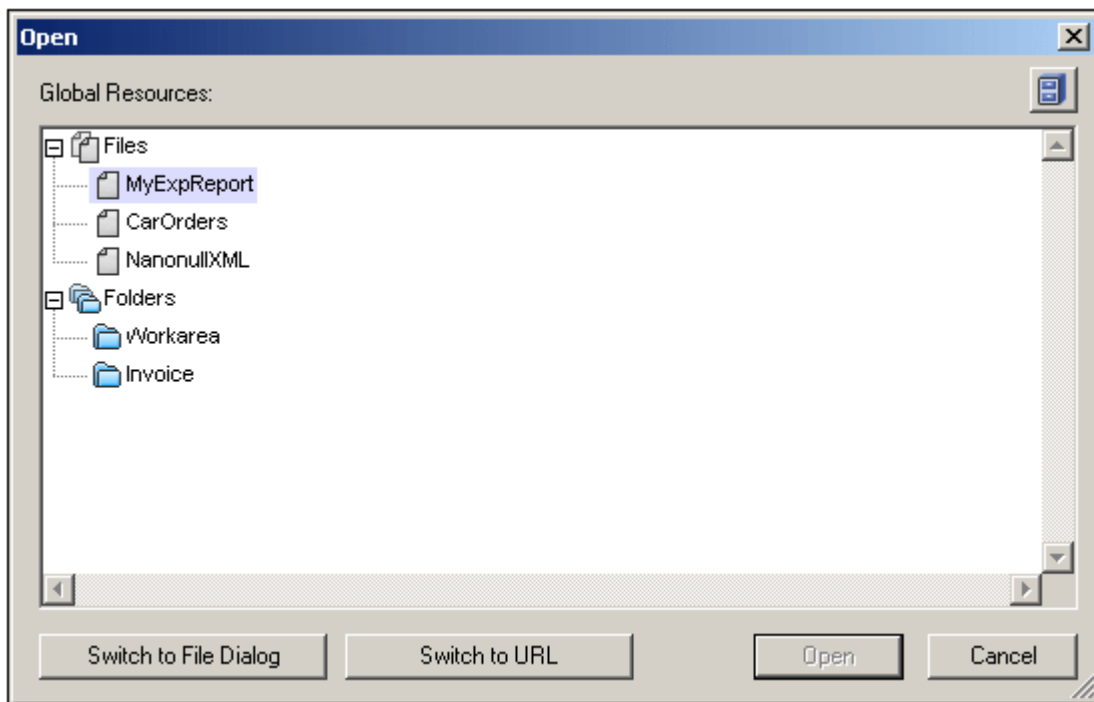
Selections that determine which resource is used

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.

14.2.1 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 browse for the required file. The path to the selected file is entered relative to the folder resource. So if a folder-type global resource were to have two configurations, each pointing to different folders, files having the same name but in different folders could be targeted via the two configurations. This could be useful for testing purposes.

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:

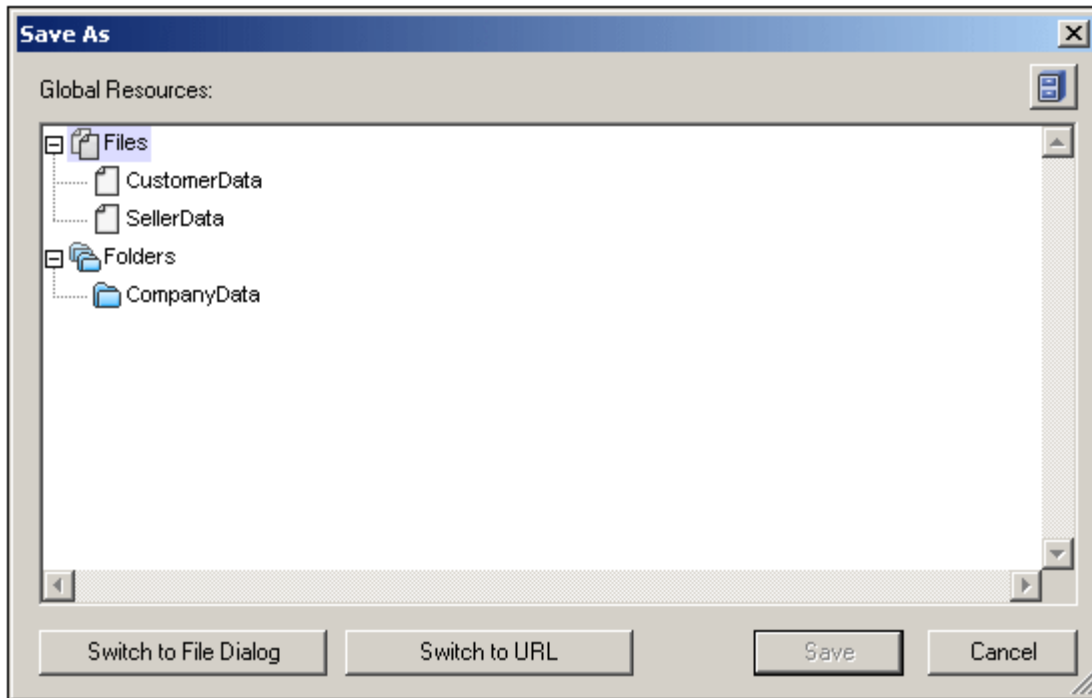
- [Opening global resources](#)
- [Saving as global resource](#)
- [Assigning files for XSLT transformations](#)
- [XSLT transformation and XQuery executions](#)
- [Assigning an SPS](#)

Opening global resources

A global resource can be opened in XMLSpy with the [File | Open \(Switch to Global Resource\)](#) command and can be edited. In the case of a file-type global resource, the file is opened directly. In the case of a folder-type global resource, an Open dialog pops up with the associated folder selected. You can then browse for the required file in descendant folders. One advantage of addressing files for editing via global resources is that related files can be saved under different configurations of a single global resource and accessed merely by changing configurations. Any editing changes would have to be saved before changing the configuration.

Saving as global resource

A newly created file can be saved as a global resource. Also, an already existing file can be opened and then saved as a global resource. When you click the **File | Save** or **File | Save As** commands, the Save dialog appears. Click the **Switch to Global Resource** button to access the available global resources (*screenshot below*), which are the aliases defined in the current Global Resources XML File.



Select an alias and click Save. If the alias is a [file alias](#), the file will be saved directly. If the alias is a [folder alias](#), a dialog will appear that prompts for the name of the file under which the file is to be saved. In either case the file will be saved to the location that was defined for the [currently active configuration](#).

Note: Each configuration points to a specific file location, which is specified in the definition of that configuration. If the file you are saving as a global resource does not have the same filetype extension as the file at the current file location of the configuration, then there might be editing and validation errors when this global resource is opened in XMLSpy. This is because XMLSpy will open the file assuming the filetype specified in the definition of the configuration.

Assigning files for XSLT transformations

XSLT files can be assigned to XML documents and XML files to XSLT documents via global resources. When the commands for assigning XSLT files ([XSL/XQuery | Assign XSL](#) and [XSL/XQuery | Assign XSL:FO](#)) and XML files ([XSL/XQuery | Assign Sample XML](#)) are clicked the assignment dialog pops up. Clicking the **Browse** button pops up the Open dialog, in which you can switch to the Open Global Resource dialog and select the required global resource. A major advantage of using a global resource to specify files for XSLT transformations is that the XSLT file (or XML file) can be changed for a transformation merely by changing the active configuration in XMLSpy; no new file assignments have to be made each time a transformation is required with a different file. When an XSLT transformation is started, it will use the file/s associated with the active configuration.

XSLT transformations and XQuery executions

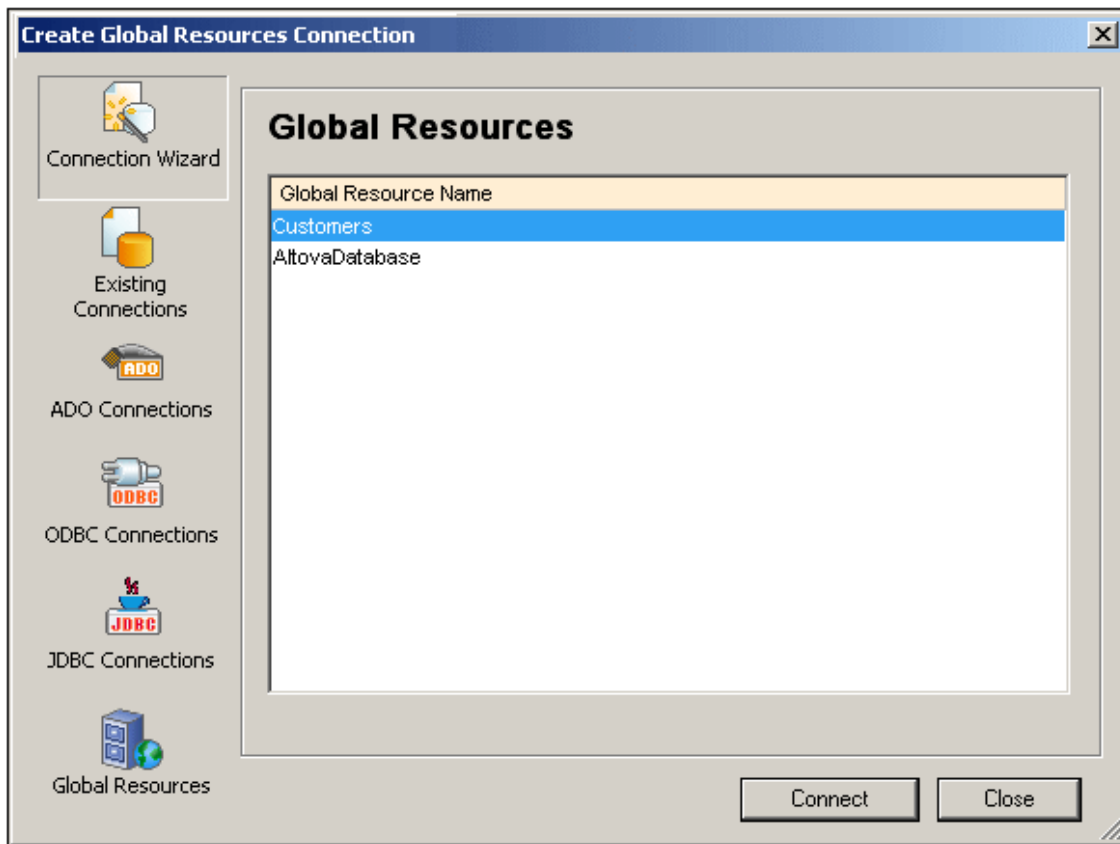
Clicking the command [XSL/XQuery | XSL Transformation](#), [XSL/XQuery | XSL:FO Transformation](#), or [XSL/XQuery | XQuery Execution](#) pops up a dialog in which you can browse for the required XSLT, XQuery, or XML file. Click the **Browse** button and then the **Switch to Global Resource** button to pop up the Open Global Resource dialog ([screenshot at top of section](#)). The file that is associated with the currently active configuration of the selected global resource is used for the transformation.

Assigning an SPS

When assigning a StyleVision stylesheet to an XML file (**Authentic | Assign StyleVision Stylesheet**), you can select a global resource to locate the stylesheet. Click the **Browse** button and then the **Switch to Global Resource** button to pop up the Open Global Resource dialog ([screenshot at top of section](#)). With a global resource selected as the assignment, the Authentic View of the XML document can be changed merely by changing the active configuration in XMLSpy.

14.2.2 Assigning Databases

When a command is executed that imports data or a data structure (as an XML Schema) from a DB into XMLSpy (for example, with the **Convert | Import Database Data** command), you can select the option to use a global resource ([screenshot below](#)). Other commands where a database-type global resource can be used are database-related commands in the menu.



When you click the Global Resources icon in the Open Database dialog, all the database-type

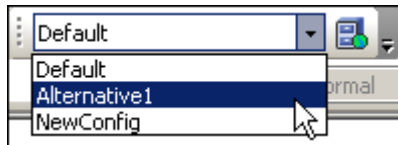
global resources that have been defined in the currently active [Global Resources XML File](#) are displayed. Select the required global resource and click **Connect**. If the selected global resource has more than one configuration, then the database resource for the currently active configuration (check **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 [Creating an XML Schema from a DB](#) and [Importing DB data](#).

14.2.3 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 an XSLT file has been assigned to an XML document via a global resource with multiple configurations. The XSLT 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 **Tools | Customize | Toolbars | Global Resources**.)

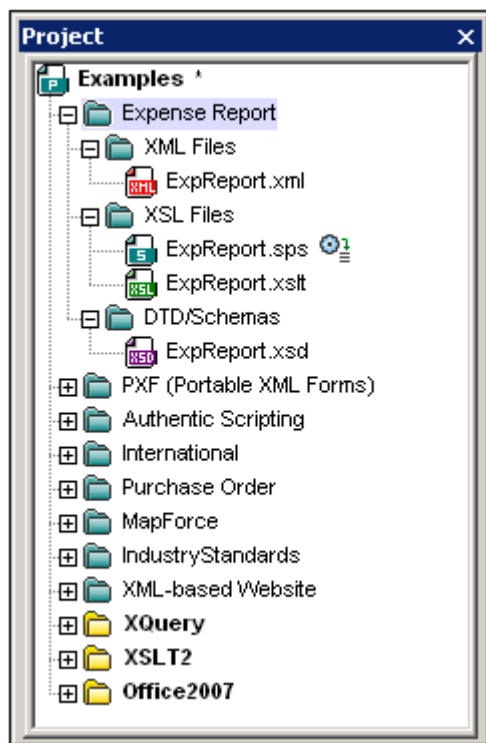


The XSLT file will be changed immediately.

In this way, by changing the active configuration, you can change source files that are assigned via a global resource.

15 Projects

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

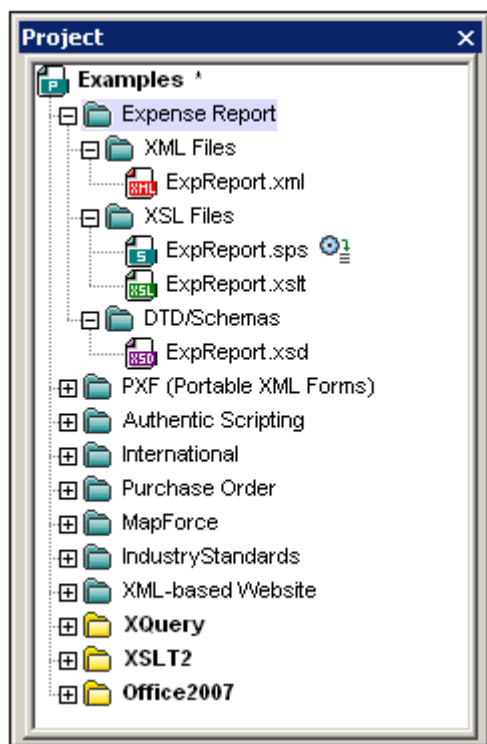


Projects thus enable you to gather together files that are used together and to access them quicker. Additionally, you can define schemas and XSLT files for individual folders, thus enabling the batch processing of files in a folder.

This section describes [how to create and edit projects](#) and [how to use projects](#).

15.1 Creating and Editing Projects

Projects are managed via the [Project Window](#) (*screenshot below*) and the [Project menu](#). One project can be open at a time in the application. The open project is displayed in the [Project Window](#).



Creating new projects, opening existing projects

A new project is created with the menu command **Project | New Project**. An existing project is opened with the menu command **Project | Open Project**. The newly opened project (whether new or existing) replaces the previously opened project in the Project Window. If the previously opened project contains unsaved changes (indicated by an asterisk next to the folder name; *see screenshot below*), you are asked whether you wish to save these changes.

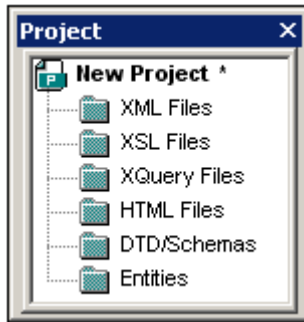
Naming and saving projects

A new project is named when you save it. A project is saved with the **Project | Save Project** command and has the `.spp` file extension. After a project has been modified, the project must be saved for the modifications to be stored. Note that a project (indicated by the top-level folder in the Project Window) can only be re-named by changing its name in Windows File Explorer; the name cannot be changed in the GUI. (The names of sub-folders, however, can be changed in the GUI.)

Project structure

A project has a tree structure of folders and files. Folders and files can be created at any level and to an unlimited depth. Do this by selecting a folder in the Project Window and then using the commands in the **Project** menu or context menu to add folders, files, or resources. Folders, files, and resources that have been added to a project can be deleted or dragged to other locations in the project tree.

When a new project is created, the default project structure organizes the project by file type (XML, XSL, etc) (see screenshot below).



File-type extensions are associated with a folder via the property definitions for that folder. When a file is added to a folder, it is automatically added to the appropriate child folder according to the file-type extension. For each folder, you can define what file-type extensions are to be associated with it.

What can be added to a project

Folder, files, and other resources can be added either to the top-level project folder or to a folder at any level in the project. There are three types of folders: (i) project folders; (ii) external folders; (iii) external web folders.

To add an object, select the relevant folder and then the required command from the **Project** menu or context menu of the selected folder. The following objects are available for addition to a project folder

- *Project folders* (green) are folders that you add to the project in order to structure the project's contents. You can define what file extensions are to be associated with a project folder (in the properties of that folder). When files are added to a folder, they are automatically added to the first child folder that has that file's extension associated with it. Consequently, when multiple files are added to a folder, they will be distributed by file extension among the child folders that have the corresponding file-extension associations.
- *External folders* (yellow) are folders in a file system. When an external folder is added to a folder, the external folder and all its files, sub-folders, and sub-folder files are included in the project. Defining file extensions on an external folder serves to filter the files available in the project.
- *External web folders* are like external folders, except that they are located on a web server and require user authentication to access. Defining file extensions on an external web folder serves to filter the files available in the project.
- *Files* can be added to a folder by selecting the folder and then using one of the three Add-File commands: (i) **Add Files**, to select the file/s via an Open dialog; (ii) **Add Active File**, to add the file that is active in the Main Window; (iii) **Add Active and Related Files**, additionally adds files related to an active XML file, for example, an XML Schema or DTD. Note that files associated by means of a processing instruction (for example, XSLT files), are not considered to be related files.
- *Global Resources* are aliases for file, folder, and database resources. How they are defined and used is described in the section on [Global Resources](#).
- *URLs* identify a resource object via a URL.
- *An Altova Scripting Project*, which is a `.asprj` file, can be assigned to an XMLSpy project. This will make macros and other scripts available to the project. How to create a Scripting Project and assign one to an XMLSpy project is described in the section, [Scripting](#).

Project properties

The properties of a folder are stored in the Properties dialog of that folder. It is accessed by first selecting the folder and then the **Properties** command in the **Project** menu or context menu (obtained by right-clicking the folder). Note that properties can be defined not only for the top-level project folder, but also for folders at various levels of the project hierarchy. The following properties of a folder can be defined and edited in the Properties dialog:

- *Folder name*: cannot be edited for the top-level project folder (for which, instead of a name, a filepath is displayed).
- *File extensions*: cannot be edited for the top-level project.
- *Validation*: specifies the DTD or XML Schema file that should be used to validate XML files in a folder.
- *Transformations*: specifies (i) the XSLT files to be used for transforming XML files in the folder, and (ii) the XML files to be transformed with XSLT files in the folder.
- *Destination files*: for the output of transformations, specifies the file extension and the folder where the files are to be saved.
- *SPS files for Authentic View*: specifies the SPS files to be used so that XML files in a folder can be viewed and edited in Authentic View.

Source control in projects

Source control systems that are compatible with Microsoft Visual Source-Safe are supported in projects. How to use this feature is described in the [User Reference section](#) of the manual.

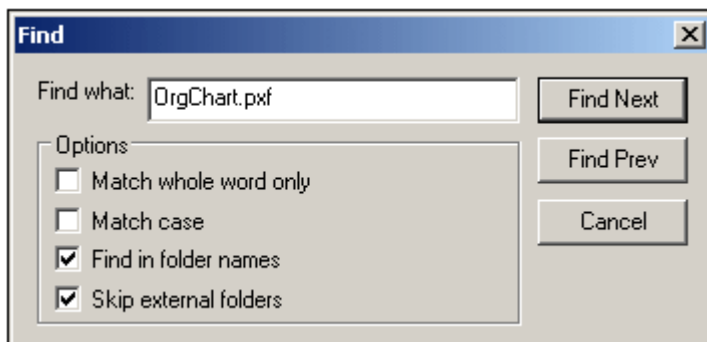
Saving projects

Any changes you make to a project, such as adding or deleting a file, or modifying a project property, must be saved with the **Save Project** command.

Find in project

You can search for project files and folders using their names or a part of their name. If the search is successful, files or folders that are located are highlighted one by one.

To start a search, activate the Project window by clicking it (or in it), then select the command **Edit | Find** (or the shortcut **Ctrl+F**). In the Find dialog that pops up (*screenshot below*) enter the text string you wish to search for and select or deselect the search options (*explained below*) according to your requirements.



The following search options are available:

- Whole-word matching is more restricted since the entire string must match an entire word in the file or folder name. In file names, the parts before and after the dot (without the dot) are each treated as a word.

- It can be specified that casing in the search string must exactly match the text string in the file or folder name.
- Folder names can be included in the search. Otherwise, only file names are searched.
- [External folders](#) can be included or excluded from the search. External folders are actual folders on the system or network, as opposed to project folders, which are created within the project and not on the system.

If the search is successful, the first matching item is highlighted in the Project sidebar. You can then browse through all the returned matching items by clicking the **Find Next** and **Find Prev** buttons in the Find dialog.

Refreshing projects

If a change is made to an external folder, this change will not be reflected in the Project Window till the project is refreshed.

15.2 Using Projects

Projects are very useful for organizing your workspace, applying settings to multiple files, and for setting up and executing batch commands. Using projects can therefore greatly help speed up and ease your work.

Benefits of using projects

The following list lists the benefits of using projects.

- Files and folders can be grouped into folders by file extension or any other desired criterion.
- Schemas and XSLT files can be assigned to a folder. This can be useful if you wish to quickly validate or transform a single XML file using different schema or XSLT files. Add the XML file to different folders and define different schemas and XSLT files for the different folders.
- Batch processing can be applied to individual folders. The commands available for batch processing are listed below.
- Output folders can be specified for transformations.

Organizing resources for quick access

Folder and file resources can be organized into a tree structure, giving you a clear overview of the various folders and files in your project, and enabling you to quickly access any and all files in a project. Simply double-click a file in the Project window to open it. You can quickly add files and folders to a project as required and delete unwanted files and folders. When you wish to work with another project, close the project currently open in the Project Window and open the required project.

Batch processing

The commands for batch processing of files in a folder, whether the top-level project folder or a folder at any other level, are **available in the context menu of that folder** (obtained by right-clicking the folder). The steps for batch processing are as follows:

1. Define the files to be used for validation or transformation in the Properties dialog of that folder.
2. Specify the folder in which the output of transformations should be saved. If no output folder is specified for a folder, the output folder of the next ancestor folder in the project tree is used.
3. Use the commands in the context menu for batch execution. If you use the corresponding commands in the XML, DTD/Schema, or XSL/XQuery menus, the command will be executed only on the document active in the Main Window, not on any project folder in the Project Window.

The following commands in the context menu of a project folder (top-level or other) are available for batch processing:

- *Well-formed check*: If any error is detected during the batch execution, it is reported in the Messages Window.
- *Validation*: If any error is detected during the batch execution, it is reported in the Messages Window.
- *Transformations*: Transformation outputs are saved to the folder specified as the output folder in the Properties dialog of that folder. If no folder is specified, the output folder of the next ancestor project folder is used. If no ancestor project folder has an output folder defined, a document window is opened and the results of each transformation is displayed successively in this document window. An XSL-FO transformation transforms an XML document or FO document to PDF.

- *Generate DTD / XML Schema*: Before the schemas are generated, you are prompted to specify an output folder. The generated schema files are saved to this folder and displayed in separate windows in the GUI.

Note: To execute batch commands use the context menu of the relevant folder in the Project Window. Do not use the commands in the XML, DTD/Schema, or XSL/XQuery menus. These commands will be executed on the document active in the Main Window.

16 File/Directory Comparisons

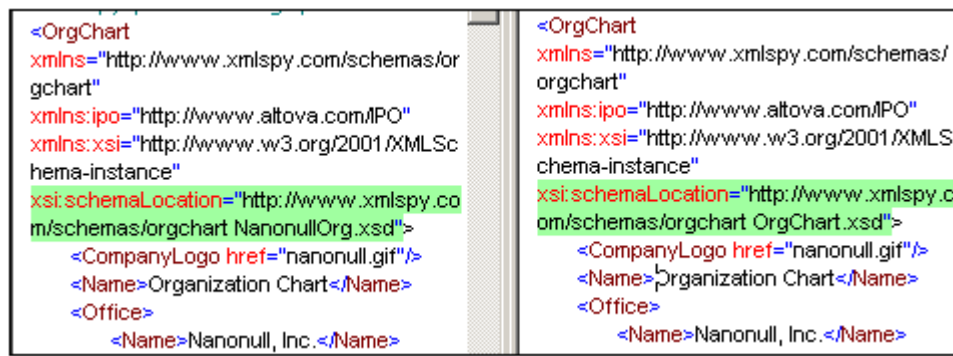
XMLSpy provides a File Comparison feature and a Directory Comparison feature that are linked to each other. File Comparisons and Directory Comparisons are started with the **Compare Open File With** and **Compare Directories** commands in the **Tools** menu, respectively. Comparison options for file comparisons can be defined in the Settings dialog, which is accessed by clicking the **Compare Options** command in the **Tools** menu.

Each of these commands is described in detail in the [User Reference](#) section. In the subsections of this section we provide an overview of the [File Comparisons](#) and [Directory Comparisons](#) mechanisms.

16.1 File Comparisons

The [File Comparisons feature](#) enables you to compare the active file with another file, which is selected via an Open File dialog or via a [global resource](#). The following points provide an overview of the mechanism. For details, see the [User Reference](#) section.

- The settings current in the [Compare Options](#) dialog when a File Compare session is started are the settings that will be active for that session.
- You can choose to compare the files as XML files (where document structure is also evaluated) or as Text files. This choice is made by selecting, in the [Settings dialog](#), either (i) Grid View or Text View (Textual Comparison Only unchecked) for XML comparisons, or (ii) Text View (Textual Comparison Only checked) for text comparisons.
- The two files appear in adjacent panes in the selected view (Grid View or Text View) and the differences are highlighted in both files (*screenshot below*).

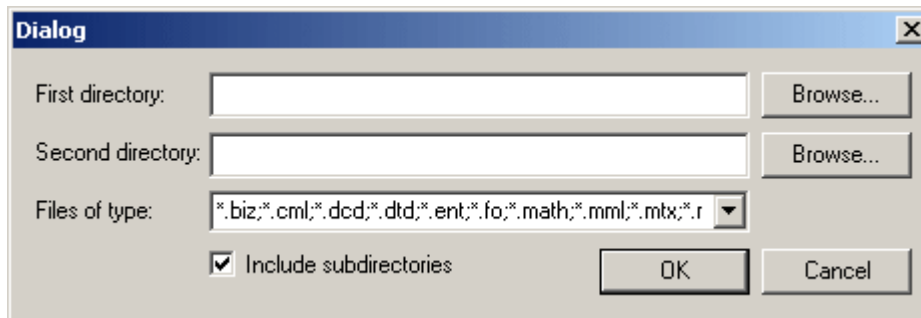


A Compare Files control window also pops up which enables you to navigate through the differences and to merge them.

The [Settings dialog](#) offers several options for specifying what aspects of the XML documents should be considered for the comparison, and what aspects ignored. For more details, see the [Compare Options](#) section in the [User Reference](#).

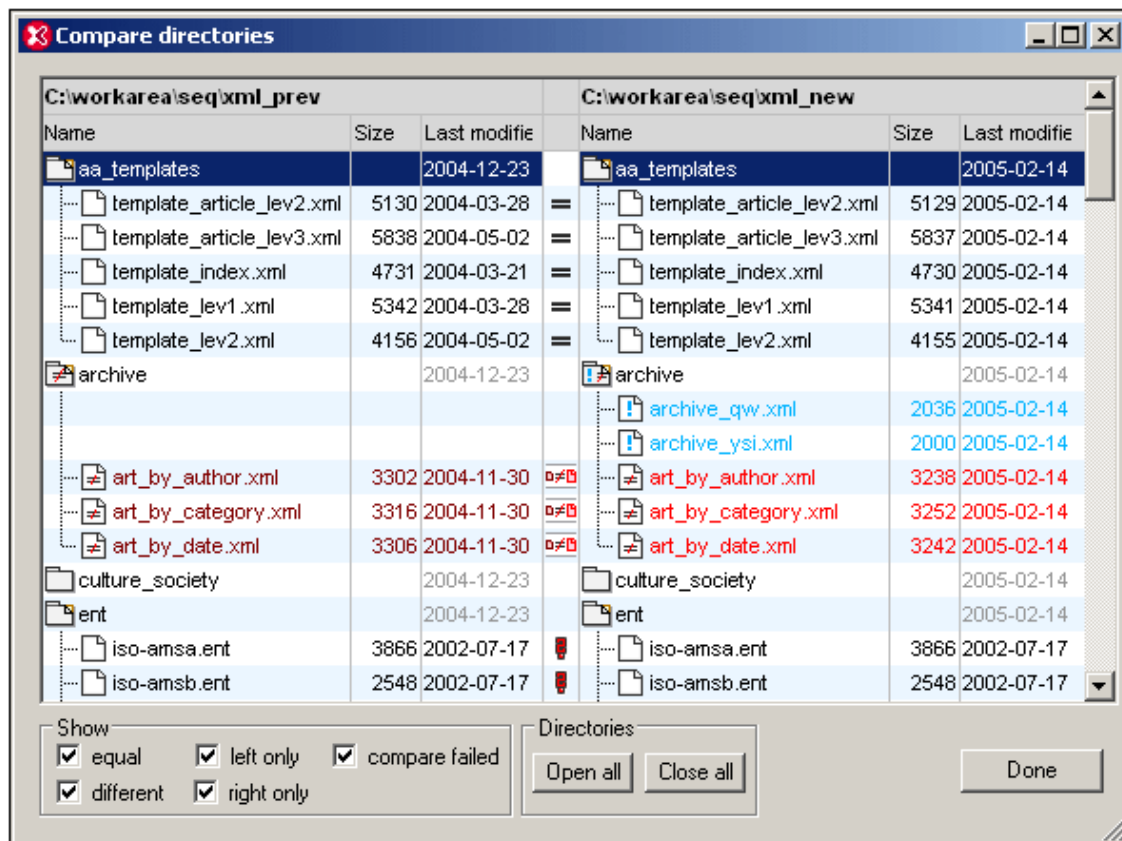
16.2 Directory Comparisons

The [Directory Comparisons feature](#) enables you to compare directories, each of which you select via separate Browse for Folder dialogs. You can also select whether sub-directories are to be compared or not, and what file types should be considered for the directory comparison.



Zip archives can also be included in the comparison by including the Zip file extension in the list of file types to evaluate.

Directories are compared to indicate missing files and whether files of the same name are different or not. The comparisons between files are based on the settings in the [Settings dialog](#). The results of the directory comparison are displayed in a separate window (*screenshot below*).



For details about how to read the symbols and manage the view in the Compare Directories window, see the description of the **Compare Directories** command in the [User Reference](#). You can then double-click a file row to directly start a file comparison.

17 Source Control

Project files can be placed under source control. A variety of source control systems are supported and Altova has tested support with a large number of drivers and source control systems. The tested systems are listed in the section, [Supported Source Control Systems](#).

Since XMLSpy implements the Microsoft Source Code Control Interface (MSSCCI) v1.1 – v1.3, multiple source control systems, including Microsoft SourceSafe and other compatible repositories, are supported.

Note: The **64-bit** version of XMLSpy automatically supports any of the supported 32-bit source control programs mentioned in this documentation. When using a 64-bit version of XMLSpy with a 32-bit source control program, the "Perform background status updates every... ms" option ([Tools | Options](#)) is automatically **grayed-out** and cannot be selected!

Overview of the Source Control feature

The mechanism for placing files in a XMLSpy project under source control is as follows:

1. In XMLSpy, a project folder containing the files to be placed under source control is created. Typically, the project folder will correspond to a local folder in which the project files are located. The path to the local folder is referred to as the local path.
2. In the source control system's database (also referred to as source control or repository), a folder is created that will contain the files to be placed under source control.
3. Project files are added to source control using the command [Project | Source Control | Add to Source Control](#).
4. Source control actions, such as checking in to, checking out from, and removing files from source control, can be carried out by using the commands in the [Project | Source Control submenu](#). The commands in this submenu are listed and described in the Project menu section of the User Reference.

Note: If you wish to change the current source control provider, this can be done in any of two ways: (i) via the Source Control options ([Tools | Options | Source Control](#)), or (ii) in the Change Source Control dialog ([Project | Source Control | Change Source Control](#)).

Note: Note that a Source Control project is not the same as an XMLSpy project. Source Control projects are directory dependent, while XMLSpy projects are logical constructions without direct directory dependence.

Registry entry and plug-ins

Microsoft has defined a Registry entry, where all Source Control-compatible programs can register themselves. This is the entry for XMLSpy:

```
HKEY_LOCAL_MACHINE\SOFTWARE\SourceCodeControlProvider\InstalledSCCProviders
```

Note that Source Control (SC) plug-ins are not automatically installed by all SC products. Please read the documentation supplied with your specific SC software for more information about plug-ins.

In this section

This section is organized as follows:

- [Supported Source Control Systems](#) lists the Source Control Systems (SCSs) that are supported by XMLSpy. Systems are listed alphabetically by server, and for each server the supported clients are also listed.
- [Installing Source Control Systems](#) contains notes about how to install the various SCSs.
- [SCSs and Altova DiffDog Differencing](#) describes how various Source Control Systems can be set up to carry out differencing with Altova DiffDog.

Menu commands

The menu commands for using Source Control Systems are in the submenu [Project | Source Control](#) and are described in the [User Reference section](#).

Resource / Speed issues

Very large source control databases might be introducing a speed/resource penalty when automatically performing background status updates.

You might be able to speed up your system by disabling (or increasing the interval of) the "Perform [background status updates every](#) xxx seconds" field in the Source Control tab accessed through **Tools | Options**.

17.1 Supported Source Control Systems

The list below shows the Source Control Servers (SCSs) supported by XMLSpy, together with their respective Source Control Client/s (SCCs). The list is organized alphabetically by SCS. Please read the notes following the list for information about support levels.

AccuRev

Version: AccuRev 4.7.0 Windows

Clients: • AccuBridge for Microsoft SCC 2008.2

Bazaar

Version: Bazaar 1.9 Windows

Clients: • Aigenta Unified SCC 1.0.6

Borland StarTeam 2008

Version: StarTeam 2008 Release 2

Clients: • Borland StarTeam Cross-Platform Client 2008 R2

Codice Software Plastic SCM

Version: Codice Software Plastic SCM Professional 2.7.127.10 (Server)

Clients: • Codice Software Plastic SCM Professional 2.7.127.10 (SCC Plugin)

Collabnet Subversion 1.5

Version: 1.5.4

Clients: • Aigenta Unified SCC 1.0.6
• PushOK SVN SCC 1.5.1.1
• PushOK SVN SCC x64 version 1.6.3.1
• TamTam SVN SCC 1.2.24

ComponentSoftware CS-RCS (PRO)

Version: ComponentSoftware CS-RCS (PRO) 5.1

Clients: • ComponentSoftware CS-RCS (PRO) 5.1

Dynamsoft SourceAnywhere for VSS

Version: Dynamsoft SourceAnywhere for VSS 5.3.2 Standard/Professional Server

Clients: • Dynamsoft SourceAnywhere for VSS 5.3.2 Client

Dynamsoft SourceAnywhere Hosted

Version: Server hosted in a Bell Data Center

Clients: • Dynamsoft SourceAnywhere Hosted Client (22252)

Dynamsoft SourceAnywhere Standalone

Version: SourceAnywhere Standalone 2.2 Server

Clients: • Dynamsoft SourceAnywhere Standalone 2.2 Client

IBM Rational ClearCase 7

Version: 7.0.1 (LT)

Clients: • IBM Rational ClearCase 7.0.1 (LT)

March-Hare CVSNT 2.5

Version: 2.5.03.2382

Clients: • Aigenta Unified SCC 1.0.6

March-Hare CVS Suite 2008

Version: Server 2008 [3321]

Clients: • Jalindi Igloo 1.0.3
• March-Hare CVS Suite Client 2008 (3321)
• PushOK CVS SCC NT 2.1.2.5
• PushOK CVS SCC x64 version 2.2.0.4
• TamTam CVS SCC 1.2.40

Mercurial

Version: Mercurial 1.0.2 for Windows

Clients: • Sergey Antonov HgSCC 1.0.1

Microsoft SourceSafe 2005

Version: 2005 with CTP

Clients: • Microsoft SourceSafe 2005 with CTP

Microsoft Visual Studio Team System 2008 Team Foundation Server

Version: 2008

Clients: • Microsoft Team Foundation Server 2008 MSSCCI Provider

Perforce 2008

Version: P4S 2008.1

Clients: • Perforce P4V 2008.1

PureCM

Version: PureCM Server 2008/3a

Clients: • PureCM Client 2008/3a

QSC Team Coherence Version Manager

Version: QSC Team Coherence Server 7.2.1.35

Clients: • QSC Team Coherence Client 7.2.1.35

Qumasoft QVCS Enterprise

Version: QVCS Enterprise 2.1.18

Clients: • Qumasoft QVCS Enterprise 2.1.18

Qumasoft QVCS Pro

Version: 3.10.18

Clients: • Qumasoft QVCS Pro 3.10.18

Reliable Software Code Co-Op

Version: Code Co-Op 5.1a

Clients: • Reliable Software Code Co-Op 5.1a

Seapine Surround SCM

Version: Surround SCM Client/Server for Windows 2009.0.0

Clients: • Seapine Surround SCM Client 2009.0.0

Serena Dimensions

Version: Dimensions Express/CM 10.1.3 for Win32 Server

Clients: • Serena Dimensions 10.1.3 for Win32 Client

Softimage Alienbrain

Version: Alienbrain Server 8.1.0.7300

Clients: • Softimage Alienbrain Essentials/Advanced Client 8.1.0.7300

SourceGear Fortress

Version: 1.1.4 Server

Clients: • SourceGear Fortress 1.1.4 Client

SourceGear SourceOffsite

Version: SourceOffsite Server 4.2.0

Clients: • SourceGear SourceOffsite Client 4.2.0 (Windows)

SourceGear Vault

Version: 4.1.4 Server

Clients: • SourceGear Vault 4.1.4 Client

VisualSVN Server 1.6

Version: 1.6.2

Clients:

- Agenta Unified SCC 1.0.6
- PushOK SVN SCC 1.5.1.1
- PushOK SVN SCC x64 version 1.6.3.1
- TamTam SVN SCC 1.2.24

Note the following:

- Altova has implemented the Microsoft Source Code Control Interface (MSSCCI) v1.1 – v1.3 in XMLSpy, and has tested support for the drivers and revision control systems listed above. It is expected that XMLSpy will continue to support these products if, and when, they are updated.
- Source Control plugins not listed in the table above, but that adhere to the MSCCI 1.1-1.3 specification, should also work together with XMLSpy.

17.2 Installing Source Control Systems

This section gives information on how to install and set up the various supported Source Control Systems.

AccuBridge for Microsoft SCC 2008.2

<http://www.accurev.com/>

1. Install AccuRev client software, run the installer and specify the server you want to connect to (hostname and port) then create a workspace.
2. Install the AccuBridge SCC provider. Extract the ZIP archive into the <AccuRev installation dir>\bin directory.
Register the AccuRev.dll and SccAcc.dll as follows:
3. Open a command prompt window (if you work with Vista, start Windows Explorer, go to C:\Windows\System32, right click and run cmd.exe "As administrator").
4. Go to the <installation AccuRev dir>\bin directory.
5. Enter the following command at the command prompt:
 - Regsvr32 AccuRev.dll
 - Regsvr32 SccAcc.dll
6. Run the SwitchScc.exe program and set AccuRev as the provider.
7. Perform a Windows log off and log in again.

Aigenta Unified SCC 1.0.6

<http://aigenta.com/products/UnifiedScc.aspx>

Requirements: source control client. Aigenta Unified SCC works with:

- subversion command line client 1.5.4 at <http://subversion.tigris.org>
- CVSNT 2.5 (client) at <http://www.cvsnt.org>
- Bazaar 1.9 Windows (and related pre-requisites) at <http://bazaar-vcs.org/> (<http://bazaar-vcs.org/WindowsInstall>)

A standard installation will work correctly with Altova products.

Borland StarTeam Cross-Platform Client 2008 R2

<http://www.borland.com/us/products/starteam>

To install the Borland StarTeam Microsoft SCC integration run the setup program and choose to install the SCC API Integration. Altova products can now connect to the repository by specifying the server address, the end point, user and password to log on, your project and working path.

Codice Software Plastic SCM Professional 2.7.127.10 (SCC Plugin)

<http://www.codicesoftware.com/xpproducts.aspx>

A standard installation will work correctly with Altova products. It is sufficient to install the "client" and the "Visual Studio SCC plug-in" components.

ComponentSoftware CS-RCS (PRO) 5.1

<http://www.componentsoftware.com/Products/RCS>

1. To install ComponentSoftware CS-RCS (PRO) start the setup and choose the option "Workstation Setup".
2. Specify your repository tree root and when the installation is finished, restart your machine as requested.
3. Use the "ComponentSoftware RCS Properties" to choose, or create, a project and to specify a work folder.

Dynamsoft SourceAnywhere for VSS 5.3.2 Client

http://www.dynamsoft.com/Products/SAW_Overview.aspx

A standard installation will work correctly with Altova products. To integrate with Altova products you do not need to install the plug-in for Adobe DreamWeaver CS3. After the installation, establish a server connection and set a working folder.

Dynamsoft SourceAnywhere Hosted Client (22252)

<http://www.dynamsoft.com/Products/SourceAnywhere-Hosting-Version-Control-Source-Control.aspx>

A standard installation will work correctly with Altova products. To integrate with the Altova products you do not need to install the plug-in for Adobe DreamWeaver CS3.

Dynamsoft SourceAnywhere Standalone 2.2 Client

<http://www.dynamsoft.com/Products/SourceAnywhere-SourceSafe-VSS.aspx>

A standard installation will work correctly with Altova products. To integrate with the Altova products you do not need to install the plug-in for Adobe DreamWeaver CS3. After the installation, establish a server connection and set a working folder.

IBM Rational ClearCase 7.0.1 (LT)

<http://www-01.ibm.com/software/awdtools/clearcase/>

To install IBM Rational ClearCase LT run the setup.

- You will be asked to update the version of the InstallShield scripting engine if it is older than version 10.5, choose "Update it if necessary". The update runs prior to the installation starting.
- Choose the default option "Enterprise deployment, create a network release area and customize it using Siteprep".

To integrate with Altova products, it is sufficient to install only the client. Check only the client check box.

- Provide a server name and the license server element(s) following the examples provided by the installer ([port@server_name](#)).
- Provide a configuration description name by editing a name you like and insert the path to a Release area. This path must specify a shared folder.
- You can create a new folder on your machine, share it, and use it as a Release Area. (In Vista, you must set the Network discovery to "on" in Network and Sharing Center to set this path.) The Release Area is now created, some files are copied into it and a shortcut is created with the name **sitedefs.lnk**.
- When all files are copied, continue by clicking the shortcut from Windows Explorer. A

- new setup will start to install the client.
- When setup starts, choose the option “Install IBM Rational ClearCase LT”.
- Keep clicking “Next”, accept the “Software License Agreement” and start the installation.

In **Vista**, the second setup could generate the internal error: 2739. In this case, start Windows Explorer and go to C:\Windows\System32.

- Right click and run “cmd.exe” “As Administrator”. A command window pops up.
- Type “regsvr32 jscript.dll”.
- Launch the setup again.

To work with files stored in ClearCase, you should create a view that points to your ClearCase project.

Jalindi Igloo 1.0.3

<http://www.jalindi.com/igloo/>

To use Jalindi Igloo with Altova products it is sufficient to run the setup to install Jalindi Igloo. Note that if you uninstall Jalindi Igloo, all other installed SCC Provider Windows registry keys (if any) are deleted as well and are not longer available.

When working with Altova products, setting the “Auto Commit” Mode is recommended.

- Auto Commit Mode is found in the advanced Source Control options.
- After defining a workspace, you can start to work.

March-Hare CVS Suite Client 2008 (3321)

<http://www.march-hare.com/cvsnt/en.asp>

A “typical” installation will work correctly with Altova products.

Mercurial

see under [Sergey Antonov HgScc 1.0.1](#)

Microsoft SourceSafe 2005 with CTP

<http://msdn.microsoft.com/en-us/vstudio/aa718670.aspx>

A standard installation of Microsoft Source Safe 2005 will work correctly with Altova products.

Microsoft Team Foundation Server 2008 MSSCCI Provider

<http://www.microsoft.com/downloads>

Requirements: Visual Studio 2008 Team Explorer, or Visual Studio 2008 with Team Explorer 2008. A standard installation will work correctly with Altova products.

Perforce P4V 2008.1

<http://www.perforce.com/>

The Perforce Visual Client (P4V) offers a choice:

- To install all client features (default behavior)
- To install only the “SCC Plug-in (P4SCC)” feature.

The default installation will work correctly with all Altova products.

If the “SCC Plug-in (P4SCC)” feature is chosen:

- Two SCC functions “Show differences” and “Source control manager” will not work.
- The “Show differences” functionality and the possibility to launch the source control manager will not work, because they rely on the non-installed features “Visual Merge Tool” and “Visual Client (P4V)” respectively.
- The differencing functionality will need 3rd party software, while the launch of the source control manager will only be possible after the explicit installation of the “Visual Client (P4V)”.
- After starting your Perforce Visual Client installation, specify your own client configuration settings (server name, Text Editing Application, User Name).
- When the installation is finished, do not forget to create a new workspace or to select an existing one.

PureCM Client 2008/3a

<http://www.purecm.com/>

A standard installation will work correctly with Altova products. After the installation, start the PureCM client to register a server.

PushOK CVS SCC NT 2.1.2.5

http://www.pushok.com/soft_cvs.php

A standard installation is sufficient for using PushOK CVS SCC NT.

- After installation is complete, make sure your copy of the CVS proxy plug-in is correctly registered.
- After defining a workspace, you can start to work.

PushOK CVS SCC x64 version 2.2.0.4

http://www.pushok.com/soft_cvs.php

A standard installation is sufficient for using PushOK CVS SCC.

- After installation is complete, make sure your copy of the CVS proxy plug-in is correctly registered.
- After defining a workspace, you can start to work.

PushOK SVN SCC 1.5.1.1

http://www.pushok.com/soft_svn.php

A standard installation of PushOK SVN SCC is sufficient for use with Altova products. When installing under Vista, it is possible that the COM library **svncom.dll** cannot be registered. In this case, finish the installation, and then register the library manually by following these steps:

1. Start a command window using the option “Run as administrator”.
2. Enter: cd “C:\Program Files\PushOK Software\SVNSCC\svn”

3. Type the command `> regsvr32 svncom.dll`.

PushOK SVN SCC x64 version 1.6.3.1

http://www.pushok.com/soft_svn.php

A standard installation of PushOK SVN SCC is sufficient for use with Altova products. When installing under Vista, it is possible that the COM library **svncom.dll** cannot be registered. In this case, finish the installation, and then register the library manually by following these steps:

1. Start a command window using the option "Run as administrator".
2. Enter: `cd "C:\Program Files\PushOK Software\SVNSCC\svn"`
3. Type the command `> regsvr32 svncom.dll`.

QSC Team Coherence Client 7.2.1.35

<http://www.teamcoherence.com>

A standard installation will work correctly with Altova products.

- If the server is installed on the client machine, a default connection is created after the client installation.
- If the server resides on a different machine, you need to change the "HOSTNAME" property in the Connection Properties dialog of Team Coherence client, to point to the relevant machine.

Qumasoft QVCS Enterprise 2.1.18

<http://www.qumasoft.com/>

Requirements: J2SE 1.5 or later <http://www.oracle.com/technetwork/java/index.html>

To install Qumasoft QVCS-Enterprise client, run the installer. If your operating system is **Vista**, you must modify the installation directory from the default value "C:\Program Files\QVCS-Enterprise Client" to "C:\QVCS-Enterprise Client". This must be done as Vista does not let applications write to the C:\Program Files area. Edit the "**setEnv.cmd**" file that resides in the installation directory so that the JAVA_HOME environment variable points to the location of your JVM.

If you work with **Vista** you might have problem when saving the file.

1. If this is the case, start Windows Explorer and go to C:\Windows\System32.
2. Right click and run "cmd.exe" "As Administrator".
3. A command window pops up.
4. Type "`cd <installation folder of the QVCS –Enterprise client>`"
5. Type "Notepad setEnv.cmd" and then edit the file and save it.
6. From the installation directory of the Qumasoft QVCS-Enterprise client run the batch file "gui.bat".
7. Add a server from the "Server menu" specifying the requested name, IP address and ports, log in and define a local workspace.

Qumasoft QVCS Pro 3.10.18

<http://www.qumasoft.com/>

To install Qumasoft QVCS-Pro run the installer.

- If your operating system is **Vista**, you must modify the installation directory from the default value “C:\Program Files\QVCSBin” to “C:\QVCSBin”. This must be done as Vista does not let applications write to the C:\Program Files area.
- After installation is finished, launch the QVCS 3.10 client, create a new user and enable **Ide integration** by selecting the submenu “Ide Integration” in the Admin menu and adding QVCS as a Version Control Tool.
- Create a project and set a workspace.

Reliable Software Code Co-Op 5.1a

http://www.relisoft.com/co_op/index.htm

A standard installation will work correctly with Altova products.

Seapine Surround SCM Client 2009.0.0

<http://www.seapine.com/surroundscm.html>

A standard installation will work correctly with Altova products. After installation, a server connection must be established.

Serena Dimensions 10.1.3 for Win32 Client

<http://www.serena.com/products/dimensions-cm/index.html>

Supported Versions: Dimensions Express/CM 10.1.3 for Win32 Client

- Perform a “Typical” installation of the Serena Dimension client.
- Specify the WEB client hostname and port number as requested.

Sergey Antonov HgSCC 1.0.1 for Mercurial SCS

http://www.newsupaplex.pp.ru/hgscs_news_eng.html

A standard installation will work correctly with Altova products.

Softimage Alienbrain Essentials/Advanced Client 8.1.0.7300

<http://www.alienbrain.com/>

- Perform a “typical” installation of the Alienbrain Client Software, then install the Alienbrain Microsoft Visual Studio Integration. To work with Altova products you do not need to install Microsoft Visual Studio.
- The first time you try to open a project from VCS, or to add a project to VCS you will be asked to enter some user settings, e.g. to specify your server and choose the project database you want to connect to.

SourceGear Fortress 1.1.4 Client

<http://www.sourcegear.com/fortress>

A standard installation of SourceGear Fortress client will work with Altova products.

SourceGear SourceOffsite Client 4.2.0 (Windows)

<http://www.sourcegear.com/sos/>

A standard installation of SourceOffsite client will work with Altova products.

SourceGear Vault 4.1.4 Client

<http://www.sourcegear.com/vault>

A standard installation of SourceGear Vault client will work correctly with Altova products.

TamTam CVS SCC 1.2.40

<http://www.daveswebsite.com/software/tamtam/>

Requirements: CVSNT 2.5 (client) at <http://www.cvsnt.org>. A standard installation will work correctly with Altova products.

- To connect to the CVS repository you need to install CVSNT.
- In the Altova product open the “Source control” Advanced options and enter the path to the **cvns.exe** executable.

TamTam SVN SCC 1.2.24

<http://www.daveswebsite.com/software/tamtamsvn/>

Requirements: subversion command line client 1.5.4 at <http://subversion.tigris.org>. A standard installation will work correctly with Altova products.

- To connect to the SVN repository you need to install the subversion command line client and specify the path to the executable **svn.exe** in the Altova product Source control options.
- After starting XMLSpy, you must register the SCC provider.

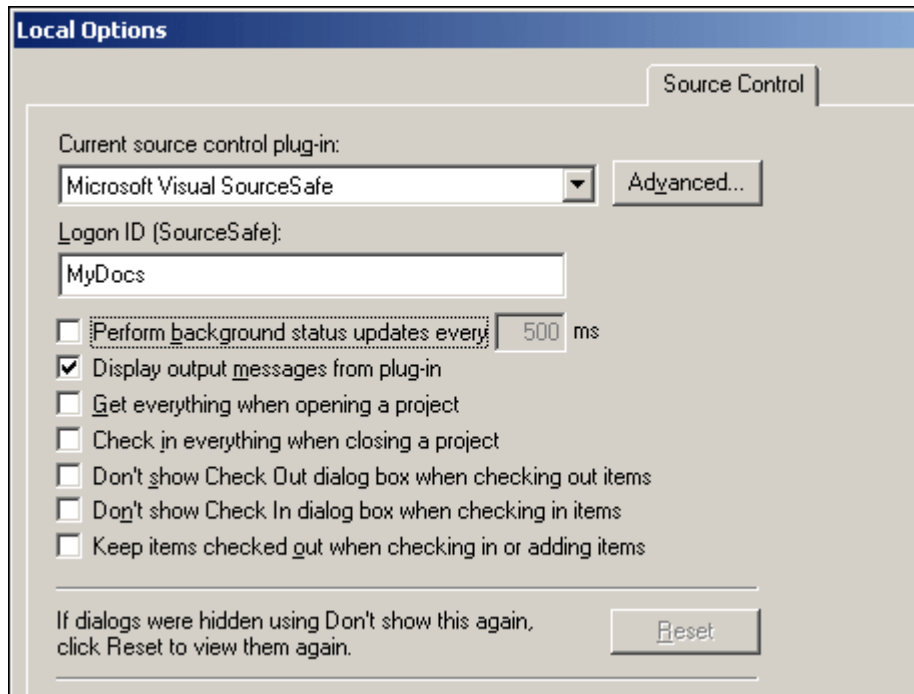
On a **Vista** machine the SCC registration could fail.

- If this is the case, use Windows explorer and browse to the directory that contains the Altova application executable.
- Right click and run the Altova executable “As Administrator”.
The SCC registration will now be successful.

17.3 SCSs and Altova DiffDog Differencing

You can configure certain Source Control Systems so that they use Altova DiffDog as their differencing tools. The systems that support this feature are listed below, together with the setup steps for each. In XMLSpy, you access the setup process via the Source Control tab of the Options dialog (**Tools | Options**, *screenshot below*).

When using a 64-bit version of XMLSpy with a 32-bit source control program, the "Perform background status updates every... ms" option (Tools | Options) is automatically **grayed-out** and cannot be selected!



The "Perform background status updates every xx ms" check box is unchecked per default, which means that status updates are not performed at all. Activate the check box and enter a value in the field, if you want to perform status updates every xx ms. For 64-bit versions using 32-bit source control plugins, this option has no effect.

In the Source Control tab, select the required Source Control System and then click the **Advanced** button. The dialog box that opens will be different for each Source Control System. For the setup process, note the following:

- If you have performed a standard installation of Altova DiffDog, the file path to the Altova DiffDog executable is:
`c:\program files\altova\diffdog2012\DiffDog.exe`
 If Altova DiffDog is installed elsewhere on your system, insert the appropriate value when the filepath is required.

Aigenta Unified SCC 1.0.6

<http://aigenta.com/products/UnifiedScc.aspx>

The following steps will integrate Altova DiffDog into Aigenta Unified SCC:

1. Click the **Advanced** button of the Source Control tab.
2. Select the *Comparison and merging tab* and specify the full DiffDog filepath as comparison tool.

Borland StarTeam Cross-Platform Client 2008 R2

<http://www.borland.com/us/products/starteam>

The following steps integrate Altova DiffDog into Borland Star Team:

1. Use the StarTeam client personal options (**Tools | Personal options | File | Alternate applications**)
2. Compare utility: Enter the DiffDog full path.
3. Compare utility options: `$file1 $file2`.

ComponentSoftware CS-RCS (PRO) 5.1

<http://www.componentsoftware.com/Products/RCS>

The following steps will integrate Altova DiffDog into ComponentSoftware CS-RCS (Pro):

1. Go to the ComponentSoftware CS-RCS Properties.
2. In the *File Types* tab, choose a file extension and edit it.
3. Enter/select the value *Custom Tool* for the "Difference Analysis Tool", and browse to insert the DiffDog full path.

Dynamsoft SourceAnywhere for VSS 5.3.2 Client

http://www.dynamsoft.com/Products/SAW_Overview.aspx

The following steps will integrate Altova DiffDog into Dynamsoft SourceAnywhere for VSS:

1. Go to the Dynamic SourceAnywhere For VSS client Options.
2. Specify the DiffDog full path as External application for diff/merge, with the arguments:
`%FIRST_FILE% "%SECOND_FILE%.`

Warning: Do not perform these settings from the Altova product options, as there is no possibility of inserting the external application parameters.

Dynamsoft SourceAnywhere Hosted Client (22252)

<http://www.dynamsoft.com/Products/SourceAnywhere-Hosting-Version-Control-Source-Control.aspx>

Dynamsoft SourceAnywhere Standalone 2.2 Client

<http://www.dynamsoft.com/Products/SourceAnywhere-SourceSafe-VSS.aspx>

The following steps will integrate Altova DiffDog into Dynamsoft SourceAnywhere Hosted and Dynamsoft SourceAnywhere Standalone:

1. Click the **Advanced** button of the Source Control tab.
2. Specify the DiffDog full path as External program application for diff/merge with arguments `%FIRST_FILE% "%SECOND_FILE%.`

Jalindi Igloo 1.0.3

<http://www.jalindi.com/igloo/>

The following steps will integrate Altova DiffDog into Jalindi Igloo:

1. Start the **Show differences** command in XMLSpy.

2. Open the **Show Differences or Merge Files** panel.
3. Set the *External Diff Command* by entering the DiffDog full file path as the External Diff EXE path.

Warning: When using the default diff editor CvsConflictEditor, you might have problems comparing files with excessively long lines. We recommended that you "pretty print" all files (particularly .ump files) before storing them in the repository. This limits the line length, thus avoiding problems with the CVSConflictEditor.

March-Hare CVS Suite Client 2008 (3321)

<http://www.march-hare.com/cvsnt/en.asp>

The following steps will integrate Altova DiffDog into Marc-Hare CVS Suite 2008:

1. Go to the TortoiseCVS Preferences and choose the Tools tab.
2. Specify the DiffDog full path as Diff application, and the parameters %1 %2 as two-way differencing parameters.

Mercurial

see under [Sergey Antonov HgScc 1.0.1](#)

Microsoft SourceSafe 2005 with CTP

<http://msdn.microsoft.com/en-us/vstudio/aa718670.aspx>

The following steps will integrate Altova DiffDog into Microsoft SourceSafe 2005:

1. Click the **Advanced** button of the Source Control tab.
2. Click the Custom Editors tab and enter C:\Program Files\Altova\DiffDog2012\DiffDog.exe %1 %2 in the Command Line field.
3. In the Operation combo box, select *File Difference*.

Microsoft Team Foundation Server 2008 MSSCCI Provider

<http://www.microsoft.com/downloads>

Requirements: Visual Studio 2008 Team Explorer or Visual Studio 2008 with Team Explorer 2008. The following steps will integrate Altova DiffDog into Microsoft Visual Studio Team System 2008 Team Foundation Server MSSCCI Provider:

1. In the manager (Visual Studio 2008 Team Explorer or Visual Studio 2008) options, configure Altova DiffDog as new user tool
2. Choose Visual Studio Team Foundation Server source as the plug-in.
3. Configure a new user tool specifying: (i) the extensions of the files you wish to compare with DiffDog; and (ii) the DiffDog full file path.

Perforce P4V 2008.1

<http://www.perforce.com/>

The following steps will integrate Altova DiffDog into Perforce 2008:

1. Click the **Advanced** button of the Source Control tab.
2. Choose the tab Diff in the Preferences panel.
3. Check as default differencing application the field "Other application" and enter the DiffDog full file path.

PushOK CVS SCC NT 2.1.2.5,
PushOK SVN SCC 1.5.1.1
PushOK CVS SCC x64 version 2.2.0.4
PushOK SVN SCC x64 version 1.6.3.1
http://www.pushok.com/soft_cv.php

The following steps will integrate Altova DiffDog into PushOK CVS NT and PushOK SVN SCC:

1. Click the **Advanced** button of the Source Control tab.
2. Choose the CVS Executables tab.
3. Select the value *External merge/compare tool* into the Diff/Merge field.
4. Insert the DiffDog full file path.
5. Edit the value `%first %second` into the "2 way diff cmd" field.

Warning: When using the default differencing editor CvsConflictEditor, you might have problems comparing files with excessively long lines. We recommended that you "pretty print" all files (particularly .ump files) before storing them in the repository. This limits the line length, thus avoiding problems with the CVSConflictEditor.

QSC Team Coherence Client 7.2.1.35

<http://www.teamcoherence.com>

The following steps will integrate Altova DiffDog into Team Coherence Version Manager:

1. Go to Team Coherence client Options "Difference Viewer".
2. Specify as the Default Difference Viewer application, the DiffDog full file path.
3. Specify as parameters: "`$LF $RF`".

Warning: It is possible that the new settings will only be applied after a Windows log off.

Qumasoft QVCS Enterprise 2.1.18

<http://www.qumasoft.com/>

The following steps will integrate Altova DiffDog into Qumasoft QVCS-Enterprise:

1. Add the Qumasoft QVCS-Enterprise installation directory to the Path environment variable.
2. Use the QVCS Enterprise User Preferences.
3. In Utilities, enable the checkbox, Use External Visual Compare Tool.
4. Specify as Visual Compare Command Line:
`<DiffDog full path> "file1Name file2Name"`

Qumasoft QVCS Pro 3.10.18

<http://www.qumasoft.com/>

The following steps will integrate Altova DiffDog into Qumasoft QVCS-Pro:

1. Use the QVCS 3.10 client preferences.
2. In Utilities, specify the DiffDog full path as the visual compare utility with parameters "`%s %s`".

Seapine Surround SCM Client 2009.0.0

<http://www.seapine.com/surroundscm.html>

The following steps will integrate Altova DiffDog into Seapine Surround SCM:

1. Go to the Surround SCM client user options (Diff/Merge) section.
2. Edit the Diff/Merge settings to compare with a selected application.
3. Enter the DiffDog full path with the parameters "%1" "%2".
4. Restart the Surround SCM client and the Altova products.

Sergey Antonov HgSCC 1.0.1

http://www.newsupaplex.pp.ru/hgscs_news_eng.html

The following steps will integrate Altova DiffDog into Mercurial:

1. Click the **Advanced** button of the Source Control tab.
2. Select differencing tool "custom", and specify the DiffDog full path.

SourceGear Fortress 1.1.4 Client

<http://www.sourcegear.com/fortress>

SourceGear Vault 4.1.4 Client

<http://www.sourcegear.com/vault>

The following steps will integrate Altova DiffDog into SourceGear Fortress and SourceGear Vault:

1. Click the **Advanced** button of the Source Control tab.
2. Set the Diff/Merge Vault options by specifying as the differencing program the DiffDog full path and using the Arguments:
`/ro1 /ro2 /title1:"%LEFT_LABEL%" /title2:"%RIGHT_LABEL%" "%LEFT_PATH%" "%RIGHT_PATH%"`

SourceGear SourceOffsite Client 4.2.0 (Windows)

<http://www.sourcegear.com/sos/>

The following steps will integrate DiffDog into SourceGear SourceOffsite:

1. Click the **Advanced** button of the Source Control tab.
2. Specify as "External Programs", "Application for comparing files" the DiffDog full path.

TamTam CVS SCC 1.2.40,**TamTam SVN SCC 1.2.24**

<http://www.daveswebsite.com/software/tamtam/>

The following steps will integrate Altova DiffDog into TamTam CVS SCC and TamTam SVN SCC:

1. Click the **Advanced** button of the Source Control tab.
2. Specify the DiffDog full file path as the external tool for Diff/Merge and Conflict.

Warning: The default differencing editor CvsConflictEditor, has problems comparing files with excessively long lines. We recommended that you "pretty print" all files (particularly .ump files) before storing them in the repository. This limits the line length, avoiding problems with the CVSConflictEditor.

18 XMLSpy in Visual Studio

XMLSpy can be integrated into the Microsoft Visual Studio IDE versions 2005, 2008, and 2010. This unifies the best of both worlds, integrating advanced XML editing capabilities with the advanced development environment of Visual Studio.

In this section, we describe:

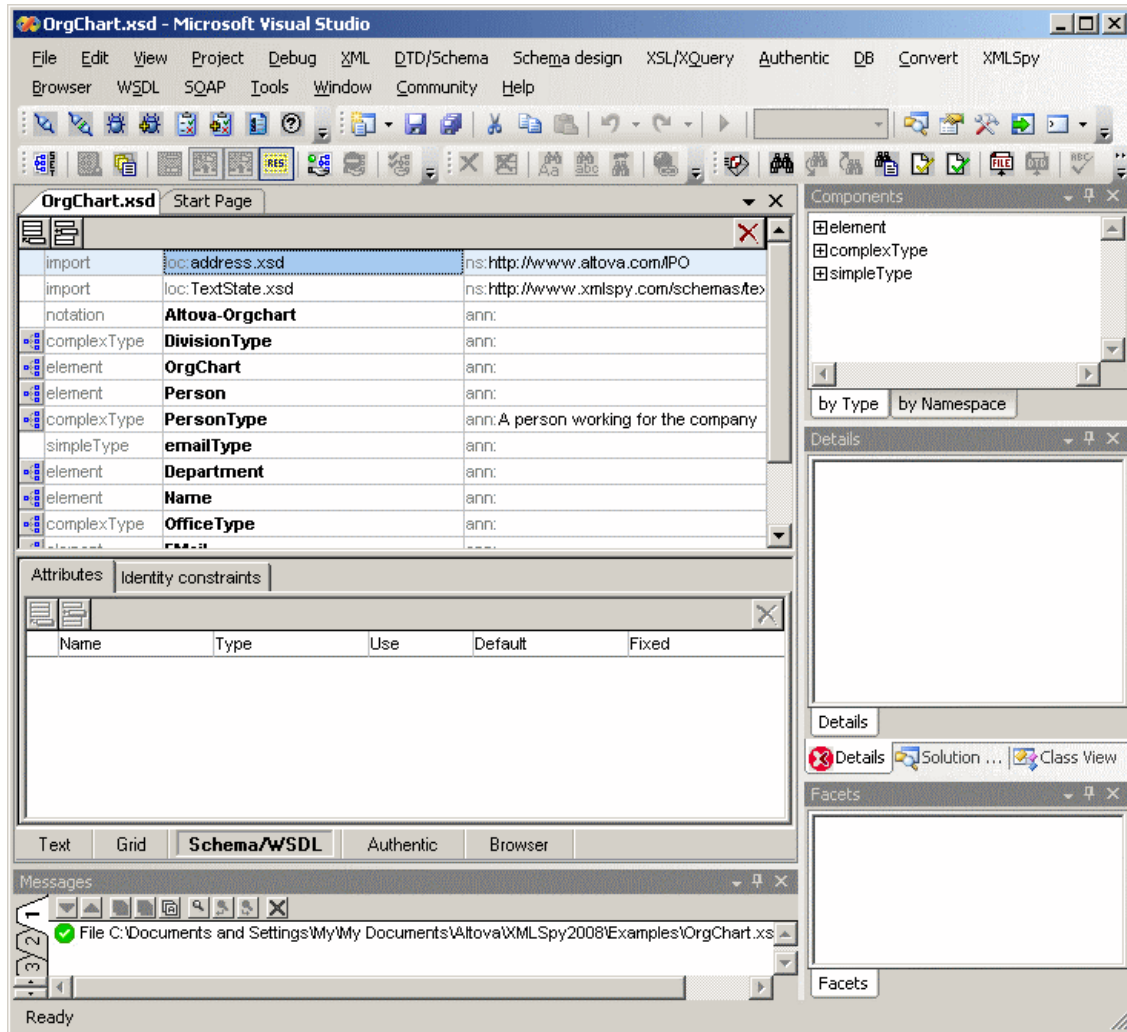
- The [broad installation process](#) and the integration of the XMLSpy plugin in Visual Studio.
- [Differences](#) between the Visual Studio version and the standalone version.
- [XMLSpy's Debuggers](#) in Visual Studio.
- [Known issues](#).

Note: The screenshots in this section are of Visual Studio version 2005. If you are using another version, there could be differences between the screenshots and your version.

18.1 Installing the XMLSpy Plugin

To install the XMLSpy Plugin for Visual Studio, you need to do the following:

- Install Microsoft Visual Studio
- Install XMLSpy (Enterprise or Professional Edition)
- Download and run the XMLSpy integration package for Microsoft Visual Studio. This package is available on the XMLSpy (Enterprise and Professional Editions) download page at www.altova.com. (**Please note:** You must use the integration package corresponding to your XMLSpy version (current version is 2012).)



Once the integration package has been installed, you will be able to use XMLSpy in the Visual Studio environment.

How to enable the plug-in

If the plug-in was not automatically enabled during the installation process, do the following:

1. Navigate to the directory where the Visual Studio IDE executable was installed, for example in `C:\Program Files\MS Visual Studio\Common7\IDE`
2. Enter the following command on the command-line `devenv.exe /setup`.
3. Wait for the process to terminate normally before starting to use the application within Visual Studio.

Note: The screenshots in this section are of Visual Studio version 2005. If you are using another version, there could be differences between the screenshots and your version.

18.2 Differences with XMLSpy Standalone

This section lists the ways in which the Visual Studio versions differ from the standalone versions of XMLSpy. The listing starts with features that are unsupported in the Visual Studio version, and continues with a listing of other ways in which the Visual Studio version differs from the standalone version.

Unsupported features in Visual Studio

The following XMLSpy features are not available in Visual Studio:

- The Scripting environment (**Tools | XMLSpy Options | Scripting**) is currently not supported.
- Separate browser window (an option in the **Tools | Options | View** tab) is not supported. This means the the Text View and Browser View are always in the same window.
- The text state icons of [Authentic View](#) are not supported.
- All Source Control functionality.
- All comparison functionality (available in the **Tools** menu of the standalone version).

Additional XMLSpy menus in Visual Studio

The following commands are specific to XMLSpy in Visual Studio:

- **View | XMLSpy Tool Windows**
- **View | XMLSpy View**
- **XMLSpy** (includes Global Resources menu items)
- **Tools | XMLSPY Options**

Entry helpers (Tool windows in Visual Studio)

The entry helpers of XMLSpy are available as Tool windows in Visual Studio. The following points about them should be noted. (For a description of entry helpers and the XMLSpy GUI, see the section, [Introduction](#).)

- You can drag entry helper windows to any position in the development environment.
- Right-clicking an entry helper tab allows you to further customize your interface. Entry helper configuration options are: dockable, hide, floating, and auto-hide.

Same functionality, different command

Some functionality of XMLSpy is available in Visual Studio under differently named commands. These are:

<i>XMLSpy</i>	<i>Visual Studio</i>	<i>Functionality</i>
File Open Switch to URL	File Open Website	Opens file from URL
Switch to URL Save	File Save XMLSpy File to URL	Saves file to URL

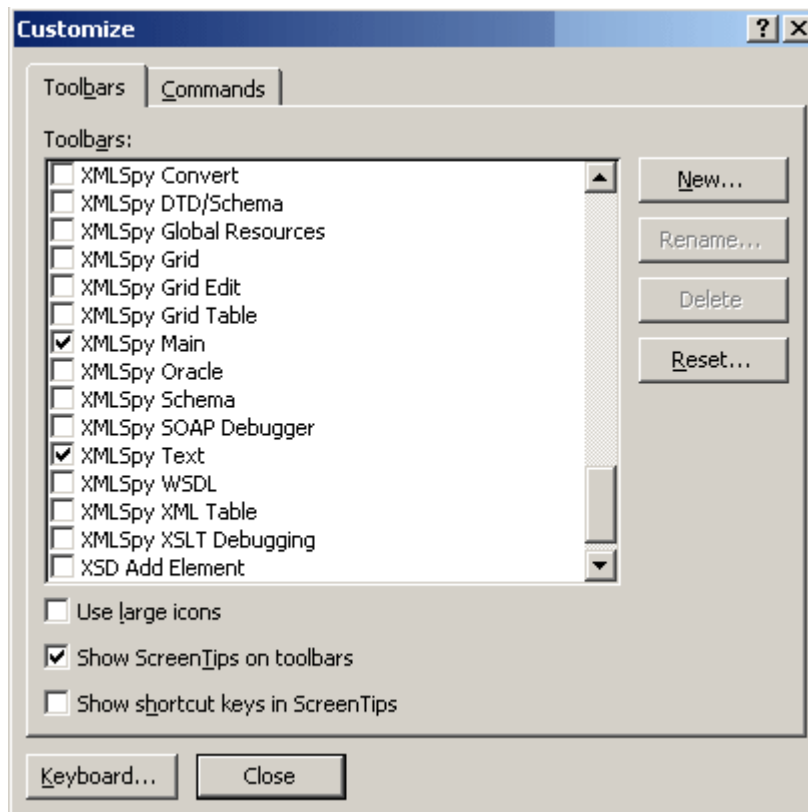
XMLSpy commands as Visual Studio commands

Some XMLSpy commands are present as Visual Studio commands in the Visual Studio GUI. These are:

- **Undo, Redo:** These Visual Studio commands affect all actions in the Visual Studio

development environment.

- **Projects:** XMLSpy projects are handled as Visual Studio projects.
- **Customize Toolbars, Customize Commands:** The Toolbars and Commands tabs (*screenshot below*) in the Customize dialog (**Tools | Customize**) contain both visual Studio commands as well as XMLSpy commands.



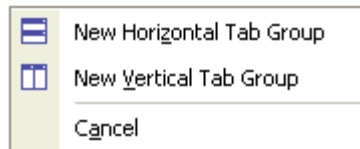
- **Views:** In the **View** menu, the two commands, **XMLSpy Tool Windows** and **XMLSpy View**, contain options to toggle on entry helper windows and other sidebars, switch between the editing views, and toggle certain editing guides on and off.
- **XMLSpy Help:** This XMLSpy menu appears as a submenu in Visual Studio's **Help** menu.

Note: The screenshots in this section are of Visual Studio version 2005. If you are using another version, there could be differences between the screenshots and your version.

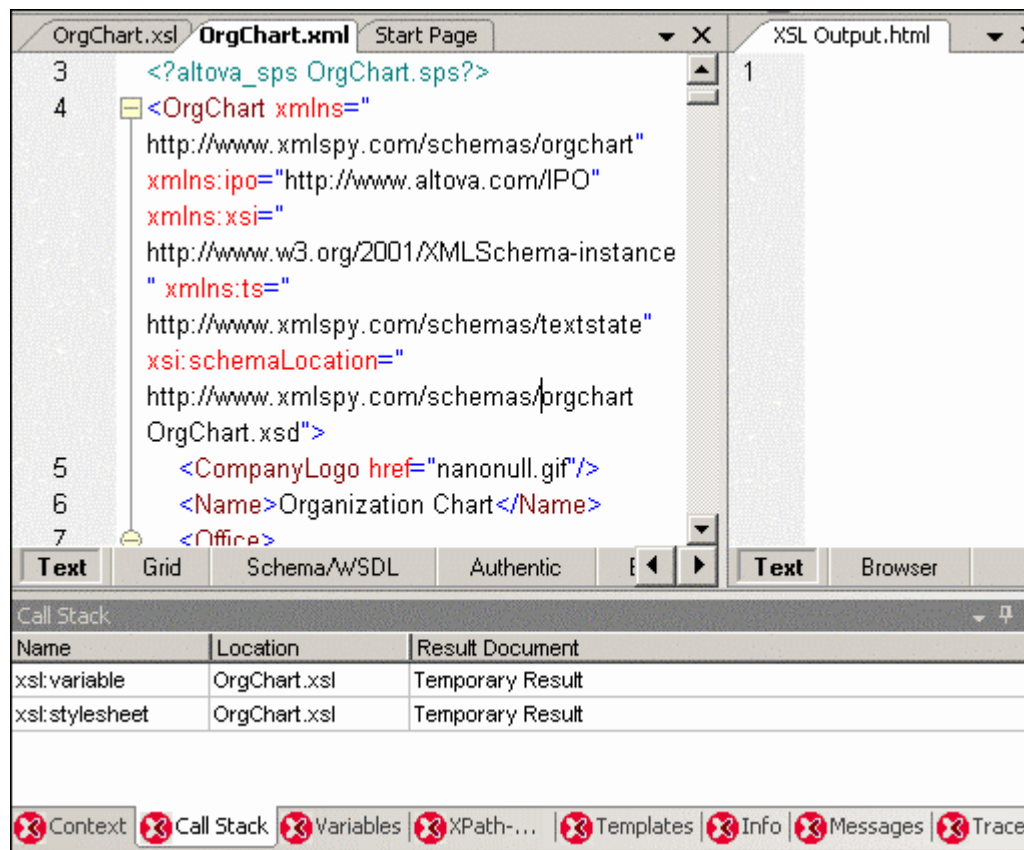
18.3 XMLSpy's Debuggers in Visual Studio

XMLSpy contains an XSLT/XQuery Debugger (*Enterprise and Professional editions*) and a SOAP Debugger (*Enterprise edition*). A debugger process involves the display of more than one file (for example, XML, XSLT, and XSLT output files), all of which are displayed in Visual Studio as a single tabbed group. To make the debugging easier to follow, you can create one or more additional tab groups in Visual Studio. Do this as follows:

1. Click the tab you wish to separate from the single tabbed group, then drag and drop it somewhere in the currently active tab. This opens a pop-up menu which allows you to define the type of tab you want to create.



2. Select **New Vertical Tab Group**. This creates a new tab containing just the selected tab (screenshot below).



Note: The screenshots in this section are of Visual Studio version 2005. If you are using another version, there could be differences between the screenshots and your version.

18.4 Known Issues

This section describes known issues relating to the different editions of MS Visual Studio and XMLSpy.

19 XMLSpy in Eclipse

Eclipse 3.x is an open source framework that integrates different types of applications delivered in the form of plugins.

The XMLSpy Plugin for Eclipse enables you to access the functionality of XMLSpy from within the Eclipse 3.5 / 3.6 / 3.7 Platform. It is available on Windows platforms. In this section, we describe [how to install](#) the XMLSpy Plugin for Eclipse and how to set up the [XMLSpy perspective](#) and [XMLSpy Debugger perspectives](#). After you have done this, components of the XMLSpy GUI and XMLSpy menu commands will be available within the Eclipse GUI.

Note:

- Source Control functionality, which is available in the standalone version, is not supported in the Eclipse version.

19.1 Installing the XMLSpy Plugin for Eclipse

Before installing the XMLSpy Plugin for Eclipse, ensure that the following are already installed:

- XMLSpy Enterprise or Professional Edition.
- Java Runtime Environment (JRE) version 1.5 or higher, which is required for Eclipse. JRE5 is recommended. See the [Eclipse website](#) for more information.
- Eclipse Platform 3.5 / 3.6 / 3.7.

After these have been installed, you can install the XMLSpy Plugin for Eclipse, which is contained in the XMLSpy Integration Package (*see below*). (**Please note:** You must use the integration package corresponding to your XMLSpy version (current version is 2012).)

Note on JRE

If, on opening a document in Eclipse, you receive the following error message:

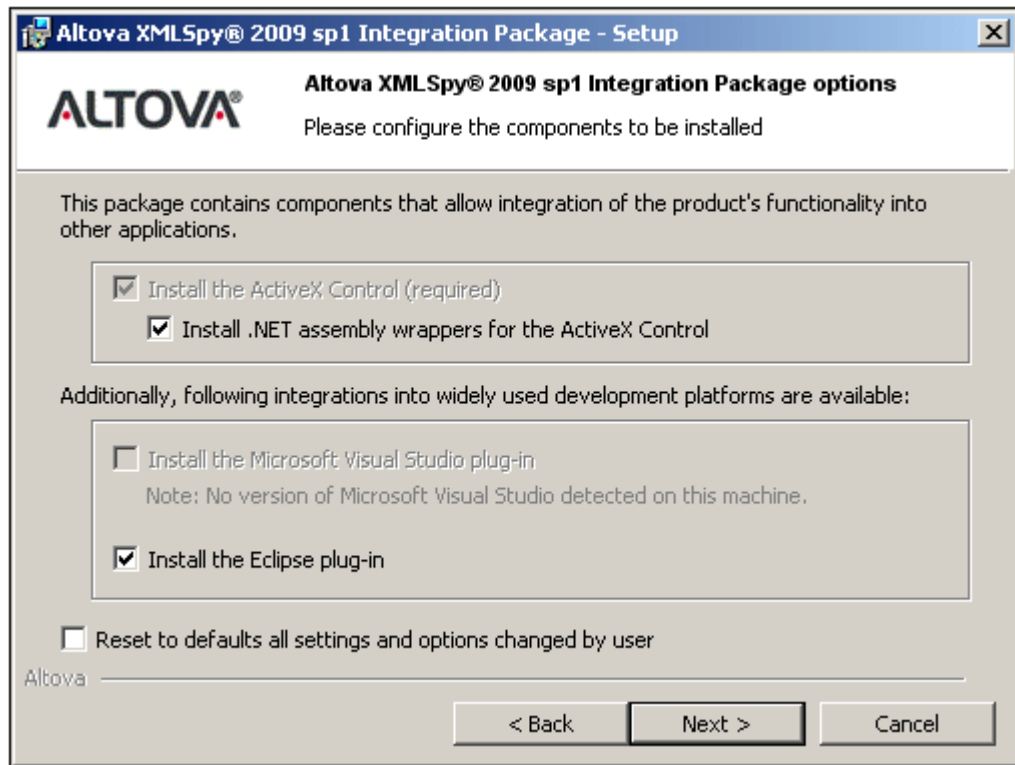
```
java.lang.UnsupportedClassVersionError: com/altova/...  
(Unsupported major.minor version 49.0)
```

it indicates that Eclipse is using an older JRE. Since Eclipse uses the `PATH` environment variable to find a `javaw.exe`, the problem can be solved by fixing the `PATH` environment variable so that a newer version is found first. Alternatively, start Eclipse with the command line parameter `-vm`, supplying the path to a `javaw.exe` of version 1.5 or higher.

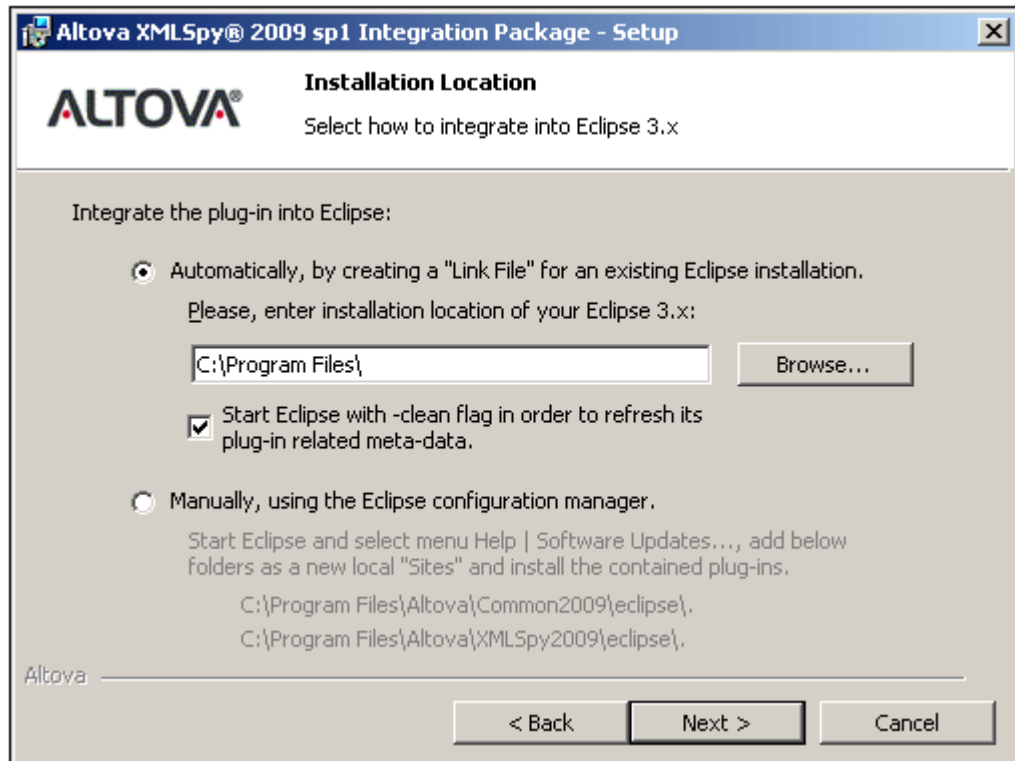
XMLSpy Integration Package

The XMLSpy Plugin for Eclipse is contained in the XMLSpy Integration Package and is installed during the installation of the XMLSpy Integration Package. Install as follows:

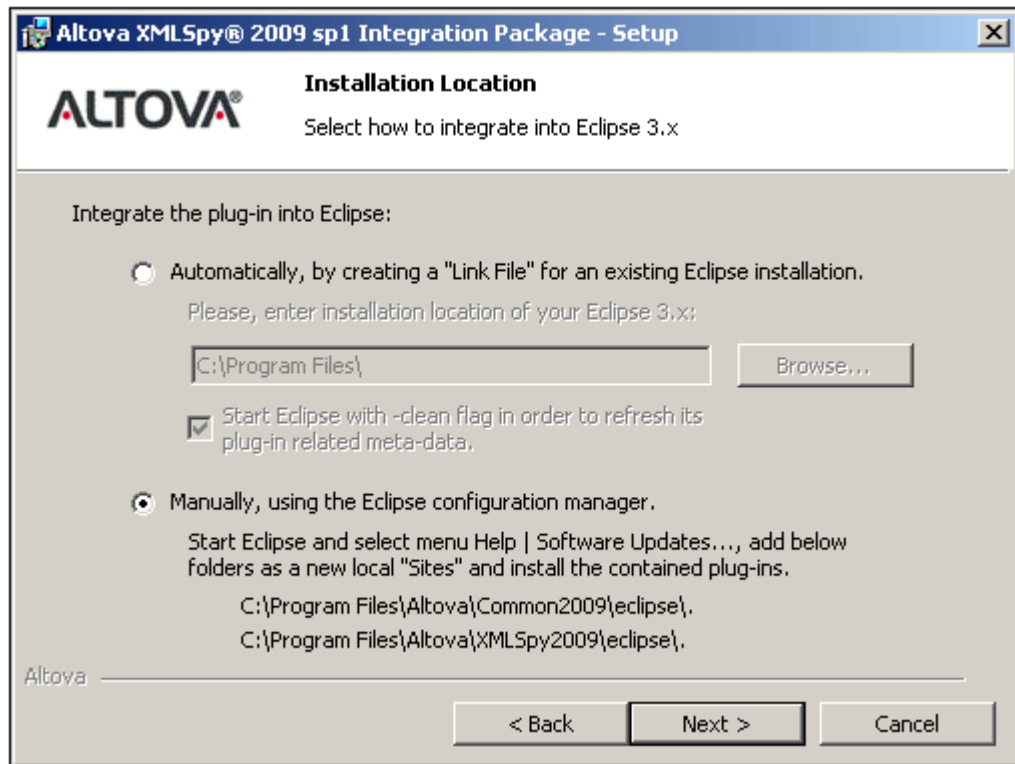
1. Ensure that XMLSpy, JRE, and Eclipse are already installed (*see above*).
2. From the [Components Download](#) page of the [Altova website](#), download and install the XMLSpy Integration Package. There are two important steps during the installation; these are described in Steps 3 and 4 below.
3. During installation of the XMLSpy Integration Package, a dialog will appear asking whether you wish to install the XMLSpy Plugin for Eclipse (*see screenshot below*). Check the option and then click **Next**.



4. In the next dialog ((Eclipse) Installation Location, *screenshot below*), you can choose whether the Install Wizard should integrate the XMLSpy Plugin into Eclipse during the installation (the "Automatic" option) or whether you will integrate the XMLSpy Plugin into Eclipse (via the Eclipse GUI) at a later time.



We recommend that you let the Installation Wizard do the integration. Do this by checking the Automatic option and then browsing for the folder in which the Eclipse executable (`eclipse.exe`) is located. Click **Next** when done. If you choose to manually integrate XMLSpy Plugin for Eclipse in Eclipse, select the Manually option (*screenshot below*). See the section below for instructions about how to manually integrate from within Eclipse.

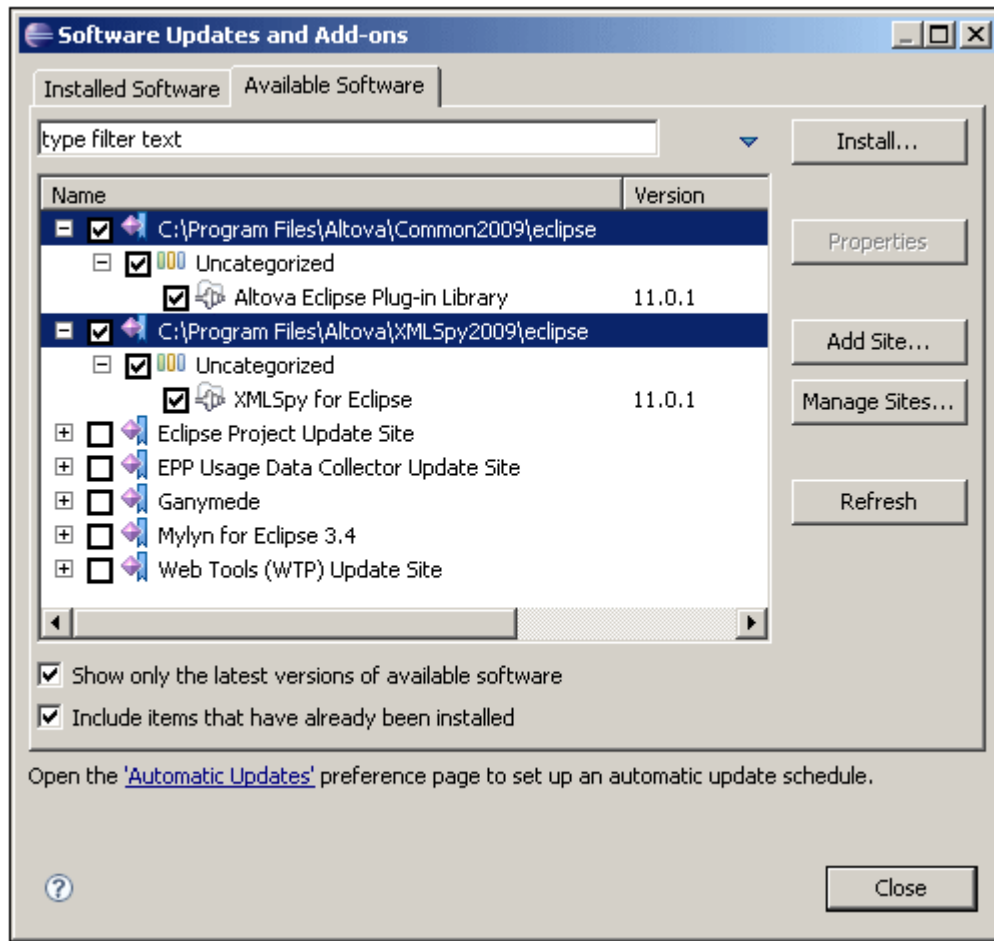


5. Complete the installation. If you set up automatic integration, the XMLSpy Plugin for Eclipse will be integrated in Eclipse and will be available when you start Eclipse the next time. Otherwise, you will have to manually integrate the plug-in (see *below*).

Manually integrating the XMLSpy plug-in in Eclipse

To manually integrate the XMLSpy Plugin for Eclipse, do the following:

1. In Eclipse, click the menu command **Help | Install New Software**.
2. In the Software Updates and Add-Ons dialog that pops up, click the Available Updates tab (*screenshot below*).
3. Click the **Add Site** button.
4. In the Add Site dialog that pops up, click the **Local** button.
5. Browse for the folder `c:\Program Files\Altova\Common2012\eclipse`, select it, and click **OK**.
6. Repeat Steps 3 to 5, this time selecting the folder `c:\Program Files\Altova\XMLSpy 2012\eclipse`.
7. The two added folders are displayed in the Available Software tab (*screenshot below*). Check the top-level check box of each folder to select the plug-ins, and click the **Install** button.



8. An Installation review dialog box allowing you to confirm that the checked items will be installed opens.
9. Click **Next** to continue. The Review License dialog opens.
10. Read the license terms and, if you accept them, click *I accept the terms....* Then click **Finish** to complete the installation.

If there are problems with the plug-in (missing icons, for example), start Eclipse with the `-clean` flag.

Currently installed version

To check the currently installed version of the XMLSpy Plugin for Eclipse, select the Eclipse menu option **Help | About Eclipse Platform**. Then select the XMLSpy icon.

19.2 XMLSpy Entry Points in Eclipse

The following entry points in Eclipse can be used to access XMLSpy functionality:

- [XMLSpy Perspective](#), which provides XMLSpy's GUI features within the Eclipse GUI.
- [XMLSpy toolbar buttons](#), which provides access to XMLSpy Help and the Create New Document functionality.

XMLSpy Perspective

In Eclipse, a perspective is a configured GUI view with attendant functionality. When the XMLSpy Plugin for Eclipse is integrated in Eclipse, a default XMLSpy perspective is automatically created. This perspective is a GUI that includes XMLSpy's GUI elements: its editing views, menus, entry helpers, and other sidebars.

When a file having a filetype associated with XMLSpy is opened (.xml, for example), this file can be edited in the XMLSpy perspective. Similarly, a file of another filetype can be opened in another perspective in Eclipse. Additionally, for any active file, you can switch the perspective, thus allowing you to edit or process that file in another environment. There are therefore two main advantage of perspectives:

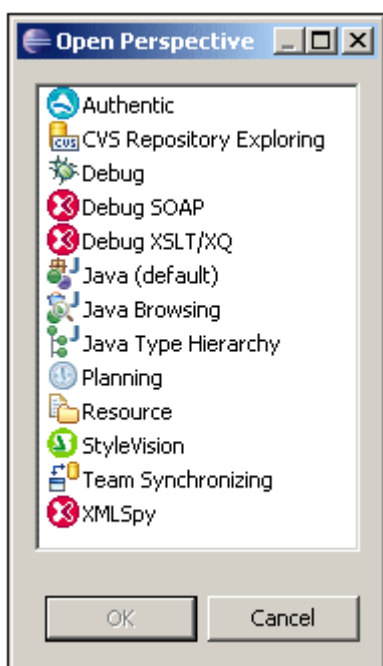
1. Being able to quickly change the working environment of the active file, and
2. Being able to switch between files without having to open a new development environment (the associated environment is available in a perspective)

Working with the XMLSpy perspective involves the following:

- Switching to the XMLSpy perspective.
- Setting preferences for the XMLSpy perspective.
- Customizing the XMLSpy perspective.

Switching to the XMLSpy perspective

In Eclipse, select the command **Window | Open Perspective | Other**. In the dialog that pops up (*screenshot below*), select **XMLSpy**, and click **OK**.



The empty window or the active document will now have the XMLSpy perspective. This is how the user switches the perspective via the menu. To access a perspective faster from another perspective, the required perspective can be listed in the **Open Perspective** submenu, above the **Other** item; this setting is in the customization dialog.

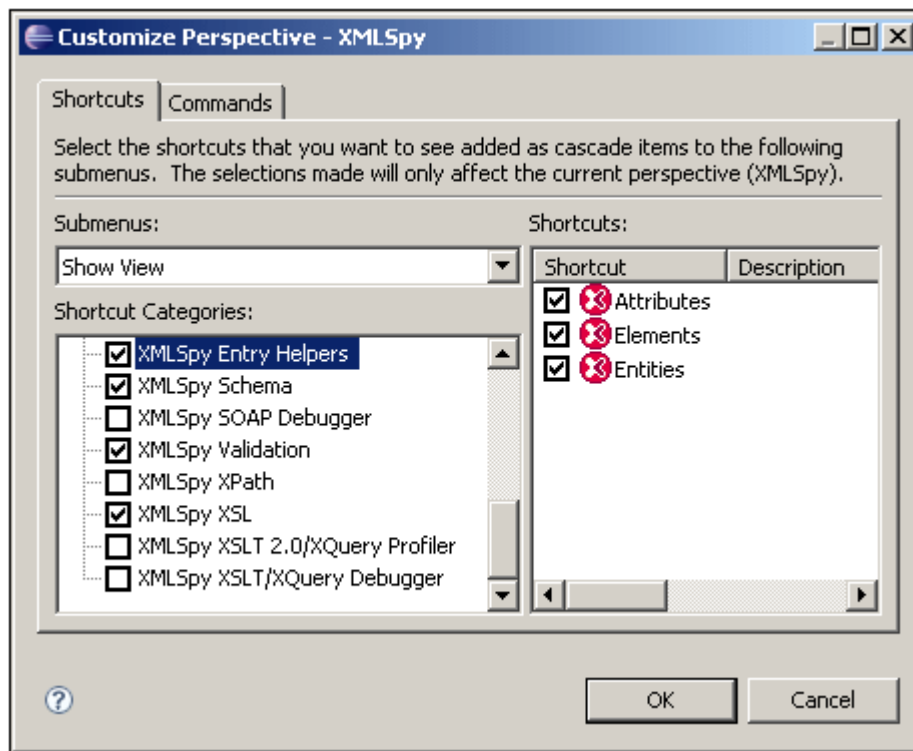
Perspectives can also be switched when a file is opened or made active. The perspective of the application associated with a file's filetype will be automatically opened when that file is opened for the first time. Before the perspective is switched, a dialog appears asking whether you wish to have the default perspective automatically associated with this filetype. Check the *Do Not Ask Again* option if you wish to associate the perspective with the filetype without having to be prompted each time a file of this filetype is opened, and then click **OK**.

Setting preferences for the XMLSpy perspective

The preferences of a perspective include: (i) a setting to automatically change the perspective when a file of an associated filetype is opened (*see above*), and (ii) options for including or excluding individual XMLSpy toolbars. To access the Preferences dialog, select the command **Window | Preferences**. In the list of perspectives in the left pane, select XMLSpy, then select the required preferences. Finish by clicking **OK**.

Customizing the XMLSpy perspective

The customization options enable you to determine what shortcuts and commands are included in the perspective. To access the Customize Perspective dialog of a perspective (*screenshot below shows dialog for the XMLSpy perspective*), make the perspective active (in this case the XMLSpy perspective), and select the command **Window | Customize Perspective**.



In the Shortcuts tab of the Customize Perspective dialog, you can set shortcuts for submenus. Select the required submenu in the Submenus combo box. Then select a shortcut category, and check the shortcuts you wish to include for the perspective.

In the Commands tab, you can add command groups. To display the commands in a command group, select the required command group from among the available command groups (displayed in the Command Groups pane). The commands in this group are displayed in a tree in the right-hand side pane, ordered hierarchically in the menu in which it will appear. If you wish to include the command group, check its check box.

Click **OK** to complete the customization and for the changes to take effect.

XMLSpy toolbar buttons

Two XMLSpy-related buttons are created automatically in the toolbar (*screenshot below*).

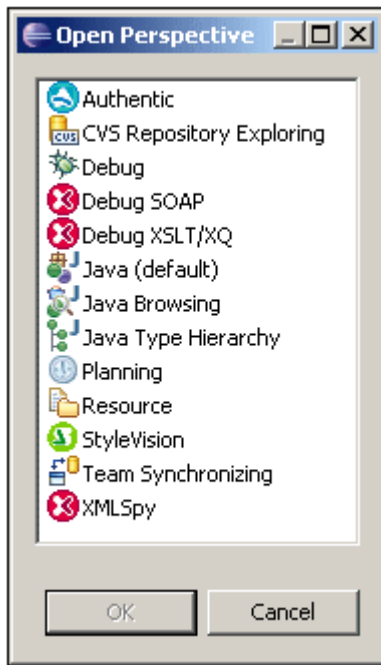


These are for: (i) opening the XMLSpy Help, and (ii) creating new XMLSpy documents .

19.3 XMLSpy's Debugger Perspectives

There are two debuggers in the Enterprise edition of XMLSpy (XSLT/XQuery and SOAP), and one debugger in the Professional edition of XMLSpy (XSLT/XQuery). Perspectives for these debuggers are available in Eclipse according to the XMLSpy edition that is currently installed.

To switch to a debugger perspective, select the command **Window | Open Perspective | Other**. In the dialog that pops up (*screenshot below*), select the debugger (for example, Debug XSLT/XQ), and click **OK**.



The empty window or the active document will now have the perspective of the selected debugger. This is how the user switches the perspective via the menu. To access a perspective faster from another perspective, the required perspective can be listed in the **Open Perspective** submenu, above the **Other** item; this setting is in the customization dialog.

For a description of how to use the debuggers, see the respective sections in this documentation: XSLT and XQuery, and WSDL and SOAP.

20 Code Generator

XMLSpy includes a built-in code generator which can automatically generate Java, C++ or C# class files from XML Schema definitions

XML is not a full programming language in that it cannot be compiled or executed as a stand-alone binary executable file; rather XML documents must be bound to an external software application or runtime environment such as a business-to-business application or Web service.

The implementation of any custom XML software application ultimately requires writing programmatic access methods within your code to create, validate, process, transform, modify or perform any in-memory operation on an XML document.

Program coding is most commonly done through the use of high-level XML processing Application Program Interfaces (API) such as Microsoft MSXML and Apache Xerces Version 2.6.0 and higher, which are freely available for various programming languages.

20.1 Introduction to code generator

In the case of XML Schemas the XMLSpy code generator's default templates automatically generate class definitions corresponding to all declared elements or complex types which redefine any complex type in your XML Schema, preserving the class derivation as defined by extensions of complex types in your XML Schema.

In the case of complex schemas which import schema components from multiple namespaces, XMLSpy preserves this information by generating the appropriate C#, or C++ namespaces or Java packages.

Additional code is implemented, such as functions which read XML files into a Document Object Model (DOM) in-memory representation, write XML files from a DOM representation back to a system file, or to convert strings to XML DOM trees and vice versa.

The output program code is expressed in C++, Java or C# programming languages.

Target Language	C++	C#	Java
Development environments	Microsoft Visual C++ 2010 Microsoft Visual C++ 2008 Microsoft Visual C++ 2005 Microsoft Visual C++ 6.0	Microsoft Visual Studio 2010 Microsoft Visual Studio 2008 Microsoft Visual Studio 2005	Apache Ant (build.xml file) Eclipse 3.4+ Borland JBuilder
XML DOM implementations	MSXML 6.0 or Apache Xerces 2.6 or later	System.Xml	JAXP
Database API (MapForce only)	ADO	ADO.NET	JDBC

C++

The C++ generated output uses either MSXML 6.0, or Apache Xerces 2.6 or later. Both MapForce and XMLSpy generate complete project and solution/workspace files for Visual C++ 6.0 or Visual Studio 2005 to 2010 directly. **.sln** and **.vcproj** files are generated in addition to the **.dsw** / **.dsp** files for Visual Studio 6.0. The generated code optionally supports MFC, e.g. by generated CString conversion methods.

Please note:

When building C++ code for Visual Studio 2005/2008/2010 and using a Xerces library precompiled for Visual C++ 6.0, a compiler setting has to be changed in all projects of the solution:

1. Select all projects in the Solution Explorer.
2. [Project] | [Properties] | [Configuration Properties] | [C/C++] | [Language]
3. Select *All Configurations*
4. Change **Treat wchar_t as Built-in Type** to **No (/Zc:wchar_t-)**

C#

The generated C# code uses the .NET XML classes (System.Xml) and can be used from any .NET capable programming language, e.g. VB.NET, Managed C++, J# or any of the several languages that target the .NET platform. Project files can be generated for Visual Studio 2005, 2008 and 2010.

Java

The generated Java output is written against the industry-standard Java API for XML Parsing

(JAXP) and includes a JBuilder project file, an Ant build file and project files for Eclipse. Java 5 (JDK 1.5) or higher is supported.

Generated output in XMLSpy:

Generated output	Location	XMLSpy
Standard libraries	Altova folder	☒
Schema wrapper libraries	Schema name folder	☒
Application		
Test Application skeleton - source code with samples	The test application must be edited for it to function. Not functional at this point.	☒

Code generator templates

Output code is completely customizable via a simple yet powerful [template language](#) which gives full control in mapping XML Schema built-in data-types to the primitive datatypes of a particular programming language.

It allows you to easily replace the underlying parsing and validating engine, customize code according to your company's writing conventions, or use different base libraries such as the Microsoft Foundation Classes (MFC) and the Standard Template Library (STL).

Additionally you can build your own templates to automate the generation of anything imaginable, for example: EJB's, WSDL files, SQL scripts, ASP or WML code.

XMLSpy's schema editor is well suited as a software modeling and prototyping tool, allowing XML applications to be rapidly prototyped at a high level in XML Schema and then automatically generated. Changes to an application's XML Schema content model can be immediately reconciled with a software implementation simply by re-running the code generator.

20.2 What's new ...

Version 2011

Contains bug fixes and enhancements.

Version 2010 R3

- Support for Microsoft Visual Studio 2010
- Support for MSXML 6.0 in generated C++ code
- Support for 64-bit targets for C++ and C# projects

Version 2010 R2

- Code generation for C++ for the [Linux platform](#)

Version 2010

- Enumeration facets from XML schemas are now available as symbolic constants in the generated classes (using 2007r3 templates).

Version 2009 sp1

- [Apache Xerces version 3.x](#) support added. (older versions starting from Xerces 2.6.x are still supported.)

Version 2009

- No new updates.

Version 2008 R2

- Support for generation of Visual Studio 2008 project files for C# and C++ has been added.

Version 2008

The new 2007 R3-style SPL templates have been further enhanced:

- It is now possible to remove single elements
- Access to schema metadata (e.g. element names, facets, enumerations, occurrence, etc.) is provided
- Complex types derived by extension are now generated as derived classes

Version 2007 R3

Code Generator has been redesigned for version 2007 release 3 to simplify usage of the generated code, reduce code volume and increase performance.

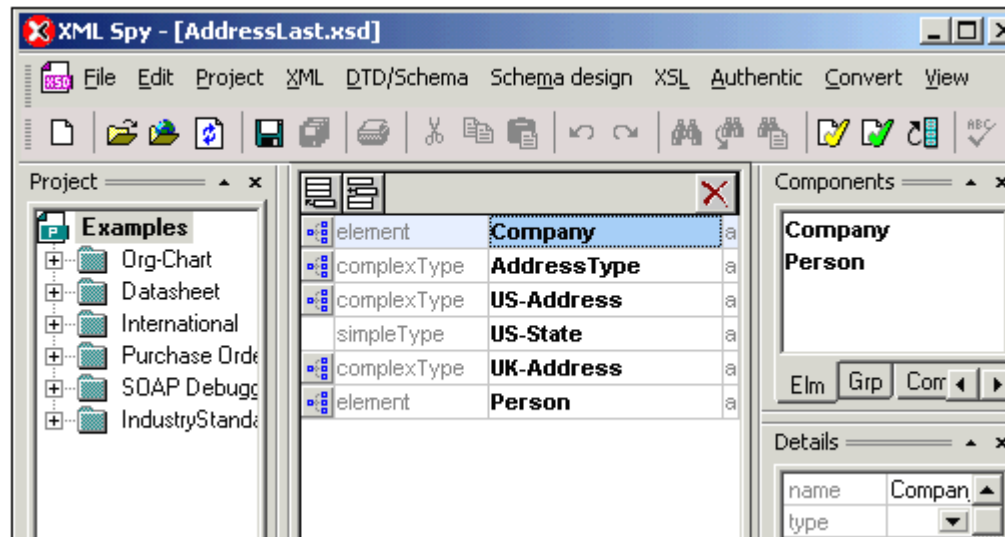
- Handling of XML documents and nodes with explicit ownership, to avoid memory leaks and to enable multi-threading
- New syntax to avoid name collisions
- New data types for simpler usage and higher performance (native types where possible, new null handling, ...)
- Attributes are no longer generated as collections
- Simple element content is now also treated like a special attribute, for consistency

- New internal object model (important for customized SPL templates)
- Compatibility mode to generate code in the style of older releases

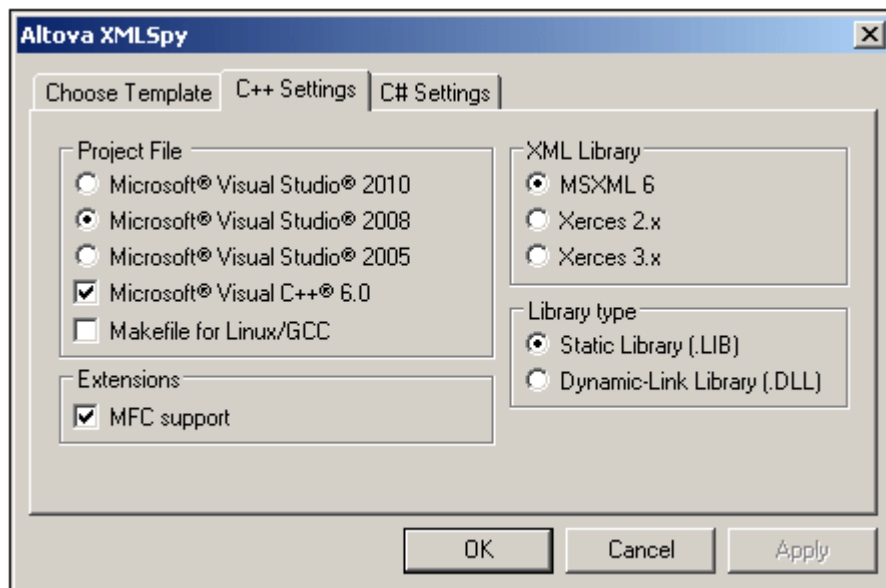
20.3 Generating source code from a schema

The following sequence shows how to generate the source code produced by the AddressLast schema file as the input.

1. Open the schema you want to generate source code for (e.g., AddressLast).



2. Select the menu item **DTD/Schema | Generate Program Code...**
3. Select the type of code you want to generate, Java, C++ or C#, (e.g., java.spl, cpp.spl or cs.spl). The C++ Settings tab allows you to further specify the code generation settings for C++, including for the Linux platform.



4. Click **OK**. The Browse for folder dialog appears.
5. Select the target folder and click **OK**.
6. You are prompted to open the newly created project in Microsoft Visual Studio. Click **Yes**.
If Java code is produced, you are prompted to open the corresponding output directory.
7. The generated workspace contains 4 projects: Altova, AltovaXML, AddressLast, and AddressLastTest. The latter is a skeleton application to test your new generated library.

Open the AddressLastTest.cpp file and write your test code into the Example() function.

Linux/GCC support

The "Makefile for Linux/GCC" option adds Makefiles to the generated code with the C++ source files being generated so that they are portable using #ifdef constructs to support different compilers and operating systems.

Note that only Xerces 3.x on Linux, with the current template version (2007r3 compatible) is supported. Enabling the "MFC support" check box will have no effect on compilation with Linux/GCC.

20.4 Using the generated code library

Overview

The library generated by XMLSpy is a DOM (W3C Document Object Model) wrapper that allows you to read, modify and create XML documents easily and safely. All data is held inside the DOM, and there are methods for extracting data from the DOM, and to update and create data into the DOM.

Compatibility Mode

Depending on the value of the setting "Generate code compatible to", the generated classes have different names, members and methods. The following description applies to **version 2007r3 and later only**. For the other settings please refer to the chapter about [using generated code compatible to old versions](#). There were no compatibility-breaking changes since version 2007r3.

Generated Libraries

When XMLSpy generates code from an XML Schema or DTD, the following libraries are generated:

Name	Purpose
Altova	Base library containing common runtime support, identical for every schema
AltovaXML	Base library containing runtime support for XML, identical for every schema
<i>YourSchema</i>	Library containing declarations generated from the input schema, named as the schema file or DTD
<i>YourSchema</i> Test	Test application skeleton (XMLSpy only)

The test application skeleton is a compilable application that calls an empty `Example()` method. You can add your test code into this method for easy and quick testing of your new generated library.

Name generation and namespaces

Generally, the code generator tries to preserve the names for generated namespaces, classes and members from the original XML Schema. Characters that are not valid in identifiers in the target language are replaced by a "_". Names that would collide with other names or reserved words are made unique by appending a number. Name generation can be influenced by changing [default settings](#) in the SPL template.

The namespaces from the XML Schema are converted to packages in Java or namespaces in C# or C++ code, using the namespace prefix from the schema as code namespace. The complete library is enclosed in a package or namespace derived from the schema file name, so you can use multiple generated libraries in one program without name conflicts.

Data Types

XML Schema has a much more elaborate data type model than Java, C# or C++. Code Generator converts the 44 XML Schema types to a couple of language-specific primitive types, or to classes delivered with the Altova library. The mapping of simple types can be configured in the SPL template.

Complex types and derived types defined in the schema are converted to classes in the generated library.

Enumeration facets from simple types are converted to symbolic constants.

Generated Classes

XMLSpy generally generates one class per type found in the XML Schema, or per element in the DTD. One additional class per library is generated to represent the document itself, which can be used for loading and saving the document.

Memory Management

A DOM tree is comprised of nodes, which are always owned by a specific DOM document - even if the node is not currently part of the document's content. All generated classes are references to the DOM nodes they represent, not values. This means that assigning an instance of a generated class does not copy the value, it only creates an additional reference to the same data.

20.4.1 Example schema

Using the generated classes

The previous sections explained how to open a schema, choose a template and generate the files you need. You then open the generated project or the corresponding directory and get down to work.

To keep things simple, we will work with a very small xsd file, the source code is shown below. Save this as **Library.xsd** if you want to get the same results as our example:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="http://www.nanonull.com/LibrarySample" xmlns:xs="
http://www.w3.org/2001/XMLSchema" targetNamespace="
http://www.nanonull.com/LibrarySample" elementFormDefault="qualified" attributeFormDefault
="unqualified">
  <xs:element name="Library">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Book" type="BookType" minOccurs="0"
maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="BookType">
    <xs:sequence>
      <xs:element name="Title" type="xs:string"/>
      <xs:element name="Author" type="xs:string" maxOccurs="unbounded"
"/>
    </xs:sequence>
    <xs:attribute name="ISBN" type="xs:string" use="required"/>
    <xs:attribute name="Variant" type="BookVariant"/>
  </xs:complexType>
  <xs:simpleType name="BookVariant">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Hardcover"/>
      <xs:enumeration value="Paperback"/>
      <xs:enumeration value="Audiobook"/>
      <xs:enumeration value="E-book"/>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

```
</xs:simpleType>
</xs:schema>
```

We will only use this schema file in this example, it will therefore be easy to figure out the differences between the different programming languages in the following code examples.

Please note that the generated classes will be named differently from xsd to xsd file. When working with your own schemas or other samples, make sure you adapt the names of the functions and variables.

20.4.2 Using the generated Java library

Please note: This topic describes code generation when using the option **Generate code compatible to V2007r3** in the code generation dialog.

Generated Packages

Name	Purpose
com.altova	Base package containing common runtime support, identical for every schema
com.altova.xml	Base package containing runtime support for XML, identical for every schema
com.YourSchema	Package containing declarations generated from the input schema, named as the schema file or DTD
com.YourSchemaTest	Test application skeleton (XMLSpy only)

DOM Implementation

The generated Java code uses JAXP (Java API for XML Processing) as the underlying DOM interface. Please note that the default DOM implementation delivered with Java 1.4 does not automatically create namespace attributes when serializing XML documents. If you encounter problems, please update to Java 5 or later.

Data Types

The default mapping of XML Schema types to Java data types is:

XML Schema	Java	Remarks
xs:string	String	
xs:boolean	boolean	
xs:decimal	java.math.BigDecimal	
xs:float, xs:double	double	
xs:integer	java.math.BigInteger	
xs:long	long	
xs:unsignedLong	java.math.BigInteger	Java does not have unsigned types.
xs:int	int	
xs:unsignedInt	long	Java does not have unsigned types.
xs:dateTime, date, time, gYearMonth, gYear, gMonthDay, gDay, gMonth	com.altova.types.DateTime	
xs:duration	com.altova.types.Duration	

xs:hexBinary and xs:base64Binary	byte[]	Encoding and decoding of binary data is done automatically.
xs:anySimpleType	string	

All XML Schema types not contained in this list are derived types, and mapped to the same Java type as their respective base type.

Generated Document Class

In addition to the classes for the types declared in the XML Schema, the document class is generated. It contains all possible root elements as members, and in addition the following methods ("Doc" stands for the name of the generated document class itself):

Method	Purpose
static <i>Doc</i> loadFromFile (String fileName)	Loads an XML document from a file
static <i>Doc</i> loadFromString (String xml)	Loads an XML document from a string
static <i>Doc</i> loadFromBinary (byte[] xml)	Loads an XML document from a byte array
void saveToFile (String fileName, boolean prettyPrint, String encoding = "UTF-8")	Saves an XML document to a file with the specified encoding. If no encoding is specified, UTF-8 is assumed.
void SaveToFile (String fileName, boolean prettyPrint, String encoding, boolean bigEndian, boolean writeBOM)	Saves an XML document to a file with the specified encoding. Byte order and Unicode byte-order mark can be specified for Unicode encodings.
String saveToString (boolean prettyPrint)	Saves an XML document to a string
byte[] saveToBinary (boolean prettyPrint, String encoding = "UTF-8")	Saves an XML document to a byte array with the specified encoding. If no encoding is specified, UTF-8 is assumed.
byte[] saveToBinary (boolean prettyPrint, String encoding, boolean bigEndian, boolean writeBOM)	Saves an XML document to a byte array with the specified encoding. Byte order and Unicode byte-order mark can be specified for Unicode encodings.
static <i>Doc</i> createDocument ()	Creates a new, empty XML document.
void setSchemaLocation (String schemaLocation)	Adds an xsi:schemaLocation or xsi:noNamespaceSchemaLocation attribute to the root element. A root element must already exist.

Generated Type Class

For each type in the schema, a class is generated that contains a member for each attribute and element of the type. The members are named the same as the attributes or elements in the original schema (in case of possible collisions, a number is appended). For simple and mixed types, **getValue()** and **setValue(string)** methods are generated. For simple types with enumeration facets, **getEnumerationValue()** and **setEnumerationValue(int)** can be used together with generated constants for each enumeration value. In addition, the method **getStaticInfo()** allows accessing schema information as **com.altova.xml.meta.SimpleType** or **com.altova.xml.meta.ComplexType** (see below).

XML Schema derivation of complexTypes by extension results in derived classes in the generated library.

Generated Member Attribute Class

For each member attribute of a type, a class with the following methods is created.

Method	Purpose
boolean exists()	Returns true if the attribute exists
void remove()	Removes the attribute from its parent element
<i>AttributeType</i> getValue()	Gets the attribute value
void setValue(AttributeType value)	Sets the attribute value
int getEnumerationValue()	Generated for enumeration types only. Returns one of the constants generated for the possible values, or Invalid if the value does not match any of the enumerated values in the schema.
void setEnumerationValue(int)	Generated for enumeration types only. Pass one of the constants generated for the possible values to this method to set the value.
com.altova.xml.meta.Attribute getInfo()	Returns an object for querying schema information (see below)

Generated Member Element Class

For each member element of a type, a class with the following methods created.

Method	Purpose
<i>MemberType</i> first()	Returns the first instance of the member element
<i>MemberType</i> at(int index)	Returns the instance of the member element
<i>MemberType</i> last()	Returns the last instance of the member element
<i>MemberType</i> append()	Creates a new element and appends it to its parent
boolean exists()	Returns true if at least one element exists
int count()	Returns the count of elements
void remove()	Deletes all occurrences of the element from its parent
void removeAt(int index)	Deletes the occurrence of the element specified by the index
java.util.Iterator iterator()	Returns an object for iterating instances of the member element.
<i>MemberType</i> getValue()	Gets the element content (only generated if element can have simple or mixed content)
void setValue(MemberType value)	Sets the element content (only generated if element can have simple or mixed content)
int getEnumerationValue()	Generated for enumeration types only. Returns one of the constants generated for the possible values, or Invalid if the value does not match any of the enumerated values in the schema.
void setEnumerationValue(int)	Generated for enumeration types only. Pass one of the constants generated for the possible values to this method to set the value.
com.altova.xml.meta.Element getInfo()	Returns an object for querying schema information (see below)

Creating XML Documents

The generated code library makes it very easy to create XML files. Let's start with an example XML file for the [example schema](#):

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<Library xsi:schemaLocation="http://www.nanonull.com/LibrarySample Library.xsd" xmlns="
http://www.nanonull.com/LibrarySample" xmlns:xsi="
http://www.w3.org/2001/XMLSchema-instance">
  <Book ISBN="0764549642" Variant="Paperback">
    <Title>The XML Spy Handbook</Title>
    <Author>Altova</Author>
  </Book>
</Library>
```

The following Java code was used to create this file. The classes Library2, LibraryType and BookType are generated out of the [schema](#) and represent the complete document, the type of the <Library> element, and the complex type BookType, which is the type of the <Book> element. Note that the automatic name de-collision renamed the Library class to Library2 to avoid a possible conflict with the library namespace name.

```
protected static void example() throws Exception {
    // create a new, empty XML document
    Library2 libDoc = Library2.createDocument();

    // create the root element <Library> and add it to the document
    LibraryType lib = libDoc.Library3.append();

    // create a new <Book> and add it to the library
    BookType book = lib.Book.append();

    // set the ISBN attribute of the book
    book.ISBN.setValue("0764549642");

    // set the Variant attribute of the book using an enumeration
    constant book.Variant.setEnumerationValue( BookVariant.EPAPERBACK );

    // add the <Title> and <Author> elements, and set values
    book.Title.append().setValue("The XML Spy Handbook");
    book.Author.append().setValue("Altova");

    // set the schema location (this is optional)
    libDoc.setSchemaLocation("Library.xsd");

    // save the XML document to a file with default encoding (UTF-8)
    // "true" causes the file to be pretty-printed.
    libDoc.saveToFile("Library1.xml", true);
}
```

Note that all elements are created by calling `append()`, and that attributes (like ISBN in the example) can simply be accessed like structure members.

Reading XML Documents

The following example reads the file we just created and demonstrates the various methods for reading XML files:

```
protected static void example() throws Exception {
    // load XML document
    Library2 libDoc = Library2.loadFromFile("Library1.xml");

    // get the first (and only) root element <Library>
    LibraryType lib = libDoc.Library3.first();

    // it is possible to check whether an element exists:
    if (!lib.Book.exists())
    {
        System.out.println("This library is empty.");
    }
}
```

```

        return;
    }

    // iteration: for each <Book>...
    for (java.util.Iterator itBook = lib.Book.iterator();
itBook.hasNext(); )
    {
        BookType book = (BookType) itBook.next();

        // output values of ISBN attribute and (first and only)
        title element
        System.out.println("ISBN: " + book.ISBN.getValue());
        System.out.println("Title: " + book.Title.first().get
Value());

        // read and compare an enumeration value
        if (book.Variant.getEnumerationValue() ==
BookVariant.EPAPERBACK)
            System.out.println("This is a paperback book.");

        // for each <Author>...
        for (java.util.Iterator itAuthor = book.Author.iterator();
itAuthor.hasNext(); )
            System.out.println("Author: " +
((com.Library.xs.stringType)itAuthor.next()).getValue());

        // alternative: use count and index
        for (int j = 0; j < book.Author.count(); ++j)
            System.out.println("Author: " + book.Author.at(j)
.getValue());
    }
}

```

Note the type of the book.Author member: xs.stringType is a generated class for any element with simple string content. Similar classes are generated for all other XML Schema types.

Of course you can also load a document, modify it by adding or deleting elements and attributes, and then save the changed file back to disk.

Error Handling

Errors are reported by exceptions. The following exception classes are defined in the namespace com.altova:

Class	Base Class	Description
SourceInstanceUnavailableException	Exception	A problem occurred while loading an XML instance.
TargetInstanceUnavailableException	Exception	A problem occurred while saving an XML instance.

In addition, the following Java exceptions are commonly used:

Class	Description
java.lang.Error	Internal program logic error (independent of input data)
java.lang.Exception	Base class for runtime errors
java.lang.IllegalArgumentException	A method was called with invalid argument values, or a type conversion failed.
java.lang.ArithmeticException	Exception thrown when a numeric type conversion fails.

Accessing Metadata

The generated library allows accessing static schema information via the following classes. The methods that return one of the metadata classes return null if the respective property does not exist.

com.altova.xml.meta.SimpleType

Method	Purpose
SimpleType getBaseType ()	Returns the base type of this type
String getLocalName ()	Returns the local name of the type
String getNamespaceURI ()	Returns the namespace URI of the type
int getMinLength ()	Returns the value of this facet
int getMaxLength ()	Returns the value of this facet
int getLength ()	Returns the value of this facet
int getTotalDigits ()	Returns the value of this facet
int getFractionDigits ()	Returns the value of this facet
String getMinInclusive ()	Returns the value of this facet
String getMinExclusive ()	Returns the value of this facet
String getMaxInclusive ()	Returns the value of this facet
String getMaxExclusive ()	Returns the value of this facet
String[] getEnumerations ()	Returns an array of all enumeration facets
String[] getPatterns ()	Returns an array of all pattern facets
int getWhitespace ()	Returns the value of the whitespace facet, which is one of: com.altova.typeinfo.WhitespaceType.Whitespace_Unknown com.altova.typeinfo.WhitespaceType.Whitespace_Preserve com.altova.typeinfo.WhitespaceType.Whitespace_Replace com.altova.typeinfo.WhitespaceType.Whitespace_Collapse

com.altova.xml.meta.ComplexType

Method	Purpose
Attribute[] GetAttributes ()	Returns a list of all attributes
Element[] GetElements ()	Returns a list of all elements
ComplexType getBaseType ()	Returns the base type of this type
String getLocalName ()	Returns the local name of the type
String getNamespaceURI ()	Returns the namespace URI of the type
SimpleType getContentType ()	Returns the simple type of the content
Element findElement (String localName, String namespaceURI)	Finds the element with the specified local name and namespace URI
Attribute findAttribute (String localName, String namespaceURI)	Finds the attribute with the specified local name and namespace URI

com.altova.xml.meta.Element

Method	Purpose
ComplexType getDataType ()	Returns the type of the element. Note that this is always a complex type even if declared as simple in the original schema. Use getContentType () of the returned object to get the simple content type.

int getMinOccurs()	Returns the minOccurs value defined in the schema
int getMaxOccurs()	Returns the maxOccurs value defined in the schema
String getLocalName()	Returns the local name of the element
String getNamespaceURI()	Returns the namespace URI of the element

com.altova.xml.meta.Attribute

Method	Purpose
SimpleType getDataType()	Returns the type of the attribute content
boolean isRequired()	Returns true if the attribute is required
String getLocalName()	Returns the local name of the attribute
String getNamespaceURI()	Returns the namespace URI of the attribute

20.4.3 Using the generated C++ library

Please note: This topic describes code generation when using the option **Generate code compatible to V2007r3** in the code generation dialog.

Generated Libraries

Name	Purpose
Altova	Base library containing common runtime support, identical for every schema
AltovaXML	Base library containing runtime support for XML (MSXML or Xerces), identical for every schema
<i>YourSchema</i>	Library containing declarations generated from the input schema, named as the schema file or DTD
<i>YourSchema</i> Test	Test application skeleton (XMLSpy only)

DOM Implementations

The generated C++ code supports either Microsoft MSXML or Apache Xerces 2.6 or higher, depending on code generation settings. The syntax for using the generated code is identical for both DOM implementations.

Character Types

The generated C++ code can be compiled with or without Unicode support. Depending on this setting, the types **string_type** and **tstring** will both be defined to **std::string** or **std::wstring**, consisting of narrow or wide characters. To use Unicode characters in your XML file that are not representable with the current 8-bit character set, Unicode support must be enabled. Please take care with the **_T()** macros. This macro ensures that string constants are stored correctly, whether you're compiling for Unicode or non-Unicode programs.

Data Types

The default mapping of XML Schema types to C++ data types is:

XML Schema	C++	Remarks
xs:string	string_type	string_type is defined as std::string or std::wstring
xs:boolean	bool	
xs:decimal	double	C++ does not have a decimal type, so double is used.
xs:float, xs:double	double	

xs:integer	__int64	xs:integer has unlimited range, mapped to int64 for efficiency reasons.
xs:nonNegativeInteger	unsigned __int64	see above
xs:int	int	
xs:unsignedInt	unsigned int	
xs:dateTime, date, time, gYearMonth, gYear, gMonthDay, gDay, gMonth	altova::DateTime	
xs:duration	altova::Duration	
xs:hexBinary and xs:base64Binary	std::vector<unsigned char>	Encoding and decoding of binary data is done automatically.
xs:anySimpleType	string_type	

All XML Schema types not contained in this list are derived types, and mapped to the same C++ type as their respective base type.

Generated Document Class

In addition to the classes for the types declared in the XML Schema, the document class is generated. It contains all possible root elements as members, and in addition the following methods ("*CDoc*" stands for the name of the generated document class itself):

Method	Purpose
static <i>CDoc</i> LoadFromFile (const string_type& fileName)	Loads an XML document from a file
static <i>CDoc</i> LoadFromString (const string_type& xml)	Loads an XML document from a string
static <i>CDoc</i> LoadFromBinary (const std::vector<unsigned char>& xml)	Loads an XML document from a byte array
void SaveToFile (const string_type& fileName, bool prettyPrint, const string_type& encoding = "UTF-8")	Saves an XML document to a file with the specified encoding. If no encoding is specified, UTF-8 is assumed.
void SaveToFile (const string_type& fileName, bool prettyPrint, const string_type& encoding, bool bigEndian, bool writeBOM)	Saves an XML document to a file with the specified encoding. Byte order and Unicode byte-order mark can be specified for Unicode encodings.
string_type SaveToString (bool prettyPrint)	Saves an XML document to a string
std::vector<unsigned char> SaveToBinary (bool prettyPrint, const string_type& encoding = "UTF-8")	Saves an XML document to a byte array with the specified encoding. If no encoding is specified, UTF-8 is assumed.
std::vector<unsigned char> SaveToBinary (bool prettyPrint, const string_type& encoding, bool bigEndian, bool writeBOM)	Saves an XML document to a byte array with the specified encoding. Byte order and Unicode byte-order mark can be specified for Unicode encodings.
static <i>CDoc</i> CreateDocument ()	Creates a new, empty XML document. Must be released using DestroyDocument ()
void DestroyDocument ()	Destroys a document. All references to the document and its nodes are invalidated. This must be called when finished working with a document.
void SetSchemaLocation (const string_type& schemaLocation)	Adds an xsi:schemaLocation or xsi:noNamespaceSchemaLocation attribute to the root element. A root element must already exist.

void SetDTDLocation (const string_type& dtdLocation)	Adds a DOCTYPE declaration with the specified system ID. A root element must already exist. This method is not supported for MSXML, since it is not possible to add a DOCTYPE declaration to a document in memory.

Generated Type Class

For each type in the schema, a class is generated that contains a member for each attribute and element of the type. The members are named the same as the attributes or elements in the original schema (in case of possible collisions, a number is appended). For simple types, assignment and conversion operators are generated. For simple types with enumeration facets, the methods **GetEnumerationValue()** and **SetEnumerationValue(int)** can be used together with generated constants for each enumeration value. In addition, the method **StaticInfo()** allows accessing of schema information as **altova::meta::SimpleType** or **altova::meta::ComplexType** (see below).

XML Schema derivation of complexTypes by extension results in derived classes in the generated library.

Generated Member Attribute Class

For each member attribute of a type, a class with the following methods is created.

Method	Purpose
bool exists ()	Returns true if the attribute exists
void remove ()	Removes the attribute from its parent element
int GetEnumerationValue ()	Generated for enumeration types only. Returns one of the constants generated for the possible values, or Invalid if the value does not match any of the enumerated values in the schema.
void SetEnumerationValue (int)	Generated for enumeration types only. Pass one of the constants generated for the possible values to this method to set the value.
altova::meta::Attribute info ()	Returns an object for querying schema information (see below)

In addition, assignment and conversion operators from/to the attribute's native type are created, so it can be used directly in assignments.

Generated Member Element Class

For each member element of a type, a class with the following methods is created.

Method	Purpose
<i>MemberType</i> first ()	Returns the first instance of the member element
<i>MemberType</i> operator[](unsigned int index)	Returns the member element specified by the index
<i>MemberType</i> last ()	Returns the last instance of the member element
<i>MemberType</i> append ()	Creates a new element and appends it to its parent
bool exists ()	Returns true if at least one element exists

unsigned int count ()	Returns the count of elements
void remove ()	Deletes all occurrences of the element from its parent
void remove (unsigned int index)	Deletes the occurrence of the element specified by the index
Iterator<MemberType> all ()	Returns an object for iterating instances of the member element
int GetEnumerationValue ()	Generated for enumeration types only. Returns one of the constants generated for the possible values, or Invalid if the value does not match any of the enumerated values in the schema.
void SetEnumerationValue (int)	Generated for enumeration types only. Pass one of the constants generated for the possible values to this method to set the value.
altova::meta::Element info ()	Returns an object for querying schema information (see below)

For simple and mixed content elements, assignment and conversion operators from/to the element's native type are created, so it can be used directly in assignments.

Creating XML Documents

The generated code library makes it very easy to create XML files. Let's start with an example XML file for the [example schema](#):

```
<?xml version="1.0" encoding="UTF-8"?>
<Library xsi:schemaLocation="http://www.nanonull.com/LibrarySample Library.xsd" xmlns="
http://www.nanonull.com/LibrarySample" xmlns:xsi="
http://www.w3.org/2001/XMLSchema-instance">
  <Book ISBN="0764549642" Variant="Paperback">
    <Title>The XML Spy Handbook</Title>
    <Author>Altova</Author>
  </Book>
</Library>
```

The following C++ code was used to create this file. The classes CLibrary, CLibraryType and CBookType are generated out of the [schema](#) and represent the complete document, the type of the <Library> element, and the complex type BookType, which is the type of the <Book> element.

```
using namespace Library;
void Example()
{
    // create a new, empty XML document
    CLibrary libDoc = CLibrary::CreateDocument();

    // create the root element <Library> and add it to the document
    CLibraryType lib = libDoc.Library2.append();

    // create a new <Book> and add it to the library
    CBookType book = lib.Book.append();

    // set the ISBN attribute of the book
    book.ISBN = _T("0764549642");

    // set the Variant attribute of the book using an enumeration constant
    book.Variant.SetEnumerationValue( CBookVariant::k_Paperback );

    // add the <Title> and <Author> elements, and set values
```

```

book.Title.append() = _T("The XML Spy Handbook");
book.Author.append() = _T("Altova");

// set the schema location (this is optional)
libDoc.SetSchemaLocation(_T("Library.xsd"));

// save the XML document to a file with default encoding (UTF-8).
"true" causes the file to be pretty-printed.
libDoc.SaveToFile(_T("Library1.xml"), true);

// destroy the document - all references to the document and any of its
nodes become invalid
libDoc.DestroyDocument();
}

```

Note that all elements are created by calling `append()`, and that attributes (like ISBN in the example) can simply be accessed like structure members. Remember to destroy the document when you are finished using it.

Reading XML Documents

The following example reads the file we just created and demonstrates the various methods for reading XML files:

```

using namespace Library;
void Example()
{
    // load XML document
    CLibrary libDoc = CLibrary::LoadFromFile(_T("Library1.xml"));

    // get the first (and only) root element <Library>
    CLibraryType lib = libDoc.Library2.first();

    // it is possible to check whether an element exists:
    if (!lib.Book.exists())
    {
        tcout << "This library is empty." << std::endl;
        return;
    }

    // iteration: for each <Book>...
    for (Iterator<CBookType> itBook = lib.Book.all(); itBook; ++itBook)
    {
        // output values of ISBN attribute and (first and only) title
        tcout << "ISBN: " << tstring(itBook->ISBN) << std::endl;
        tcout << "Title: " << tstring(itBook->Title.first()) << std::endl;

        // read and compare an enumeration value
        if (itBook->Variant.GetEnumerationValue() ==
            CBookVariant::k_Paperback)
            tcout << "This is a paperback book." << std::endl;

        // for each <Author>...
        for (CBookType::Author::iterator itAuthor = itBook->Author.all(); itAuthor; ++itAuthor)
            tcout << "Author: " << tstring(itAuthor) << std::endl;

        // alternative: use count and index
        for (unsigned int j = 0; j < itBook->Author.count(); ++j)
            tcout << "Author: " << tstring(itBook->Author[j]) << std::endl;
    }
}

```

```

    // destroy the document - all references to the document and any of its
    nodes become invalid
    libDoc.DestroyDocument();
}

```

To access the contents of attributes and elements as strings for the `cout <<` operator, an explicit cast to `tstring` is required.

Of course you can also load a document, modify it by adding or deleting elements and attributes, and then save the changed file back to disk.

Error Handling

Errors are reported by exceptions. The following exception classes are defined in the namespace `altova`:

Class	Base Class	Description
Error	<code>std::logic_error</code>	Internal program logic error (independent of input data)
Exception	<code>std::runtime_error</code>	Base class for runtime errors
InvalidArgumentsException	Exception	A method was called with invalid argument values.
ConversionException	Exception	Exception thrown when a type conversion fails
StringParseException	ConversionException	A value in the lexical space cannot be converted to value space.
ValueNotRepresentableException	ConversionException	A value in the value space cannot be converted to lexical space.
OutOfRangeException	ConversionException	A source value cannot be represented in target domain.
InvalidOperationException	Exception	An operation was attempted that is not valid in the given context.
DataSourceUnavailableException	Exception	A problem occurred while loading an XML instance.
DataTargetUnavailableException	Exception	A problem occurred while saving an XML instance.

All exception classes contain a message text and a pointer to a possible inner exception.

Method	Purpose
<code>string type message()</code>	Returns a textual description of the exception
<code>std::exception inner()</code>	Returns the exception that caused this exception, if available, or NULL

Accessing Metadata

The generated library allows accessing static schema information via the following classes. All methods are declared as `const`. The methods that return one of the metadata classes return a NULL object if the respective property does not exist.

`altova::meta::SimpleType`

Method	Purpose
<code>operator bool()</code>	Returns true if this is not the NULL SimpleType
<code>operator !()</code>	Returns true if this is the NULL SimpleType
<code>SimpleType GetBaseType()</code>	Returns the base type of this type
<code>string_type GetLocalName()</code>	Returns the local name of the type

string_type GetNamespaceURI()	Returns the namespace URI of the type
unsigned int GetMinLength()	Returns the value of this facet
unsigned int GetMaxLength()	Returns the value of this facet
unsigned int GetLength()	Returns the value of this facet
unsigned int GetTotalDigits()	Returns the value of this facet
unsigned int GetFractionDigits()	Returns the value of this facet
string_type GetMinInclusive()	Returns the value of this facet
string_type GetMinExclusive()	Returns the value of this facet
string_type GetMaxInclusive()	Returns the value of this facet
string_type GetMaxExclusive()	Returns the value of this facet
std::vector<string_type> GetEnumerations()	Returns a list of all enumeration facets
std::vector<string_type> GetPatterns()	Returns a list of all pattern facets
WhitespaceType GetWhitespace()	Returns the value of the whitespace facet, which is one of: Whitespace_Unknown Whitespace_Preserve Whitespace_Replace Whitespace_Collapse

altova::meta::ComplexType

Method	Purpose
operator bool()	Returns true if this is not the NULL ComplexType
operator !()	Returns true if this is the NULL ComplexType
std::vector<Attribute> GetAttributes()	Returns a list of all attributes
std::vector<Element> GetElements()	Returns a list of all elements
ComplexType GetBaseType()	Returns the base type of this type
string_type GetLocalName()	Returns the local name of the type
string_type GetNamespaceURI()	Returns the namespace URI of the type
SimpleType GetContentType()	Returns the simple type of the content
Element FindElement (const char_type* localName, const char_type* namespaceURI)	Finds the element with the specified local name and namespace URI
Attribute FindAttribute (const char_type* localName, const char_type* namespaceURI)	Finds the attribute with the specified local name and namespace URI

altova::meta::Element

Method	Purpose
operator bool()	Returns true if this is not the NULL Element
operator !()	Returns true if this is the NULL Element
ComplexType GetDataType()	Returns the type of the element. Note that this is always a complex type even if declared as simple in the original schema. Use GetContentType() of the returned object to get the simple content type.
unsigned int GetMinOccurs()	Returns the minOccurs value defined in the schema
unsigned int GetMaxOccurs()	Returns the maxOccurs value defined in the schema
string_type GetLocalName()	Returns the local name of the element
string_type GetNamespaceURI()	Returns the namespace URI of the element

altova::meta::Attribute

Method	Purpose

operator bool()	Returns true if this is not the NULL Attribute
operator !()	Returns true if this is the NULL Attribute
SimpleType GetDataType()	Returns the type of the attribute content
bool IsRequired()	Returns true if the attribute is required
string_type GetLocalName()	Returns the local name of the attribute
string_type GetNamespaceURI()	Returns the namespace URI of the attribute

20.4.4 Using the generated C# library

Please note: This topic describes code generation when using the option **Generate code compatible to V2007r3** in the code generation dialog.

Generated Libraries

Name	Purpose
Altova	Base library containing common runtime support, identical for every schema
AltovaXML	Base library containing runtime support for XML, identical for every schema
YourSchema	Library containing declarations generated from the input schema, named as the schema file or DTD
YourSchema Test	Test application skeleton (XMLSpy only)

DOM Implementation

The generated C# code uses the .NET standard System.XML library as the underlying DOM implementation.

Data Types

The default mapping of XML Schema types to C# data types is:

XML Schema	C#	Remarks
xs:string	string	
xs:boolean	bool	
xs:decimal	decimal	xs:decimal has unlimited range and precision, mapped to decimal for efficiency reasons.
xs:float, xs:double	double	
xs:long	long	
xs:unsignedLong	ulong	
xs:int	int	
xs:unsignedInt	uint	
xs:dateTime, date, time, gYearMonth, gYear, gMonthDay, gDay, gMonth	Altova.Types.Date Time	
xs:duration	Altova.Types.Dura tion	
xs:hexBinary and xs:base64Binary	byte[]	Encoding and decoding of binary data is done automatically.
xs:anySimpleType	string	

All XML Schema types not contained in this list are derived types, and mapped to the same C# type as their respective base type.

Generated Document Class

In addition to the classes for the types declared in the XML Schema, the document class is generated. It contains all possible root elements as members, and in addition the following methods ("Doc" stands for the name of the generated document class itself):

Method	Purpose
static <i>Doc</i> LoadFromFile (string fileName)	Loads an XML document from a file
static <i>Doc</i> LoadFromString (string xml)	Loads an XML document from a string
static <i>Doc</i> LoadFromBinary (byte[] xml)	Loads an XML document from a byte array
void SaveToFile (string fileName, bool prettyPrint, string encoding = "UTF-8")	Saves an XML document to a file with the specified encoding. If no encoding is specified, UTF-8 is assumed.
string SaveToString (bool prettyPrint)	Saves an XML document to a string
byte[] SaveToBinary (bool prettyPrint, string encoding = "UTF-8")	Saves an XML document to a byte array with the specified encoding. If no encoding is specified, UTF-8 is assumed.
static <i>Doc</i> CreateDocument ()	Creates a new, empty XML document
static <i>Doc</i> CreateDocument (string encoding)	Creates a new, empty XML document, with encoding of type "encoding".
void SetSchemaLocation (string schemaLocation)	Adds an xsi:schemaLocation or xsi:noNamespaceSchemaLocation attribute to the root element. A root element must already exist.
void SetDTDLocation (string dtdLocation)	Adds a DOCTYPE declaration with the specified system ID. A root element must already exist.

Generated Type Class

For each type in the schema, a class is generated that contains a member for each attribute and element of the type. The members are named the same as the attributes or elements in the original schema (in case of possible collisions, a number is appended). For simple and mixed types, a **Value** property is generated to access the text content. For simple types with enumeration facets, the **EnumerationValue** property can be used together with generated constants for each enumeration value. In addition, the property **StaticInfo**() allows accessing of schema information as **Altova.Xml.Meta.SimpleType** or **Altova.Xml.Meta.ComplexType** (see below).

XML Schema derivation of complexTypes by extension results in derived classes in the generated library.

Generated Member Attribute Class

For each member attribute of a type, a class with the following methods or properties is created.

Method/Property	Purpose
bool Exists ()	Returns true if the attribute exists
void Remove ()	Removes the attribute from its parent element
<i>AttributeType</i> Value	Sets or gets the attribute value
<i>int</i> EnumerationValue	Generated for enumeration types only. Sets or gets the attribute value using one of the constants generated for the possible values, reading returns Invalid if the value does not match any of the enumerated values in the schema.

Altova.Xml.Meta.Attribute Info	Returns an object for querying schema information (see below)

Generated Member Element Class

For each member element of a type, a class with the following methods or properties is created. The class implements the standard System.Collections.IEnumerable interface, so it can be used with the *foreach* statement.

Method/Property	Purpose
<i>MemberType</i> First	Returns the first instance of the member element
<i>MemberType</i> this[int index]	Returns the member element specified by the index
<i>MemberType</i> At (int index)	Returns the member element specified by the index
<i>MemberType</i> Last	Returns the last instance of the member element
<i>MemberType</i> Append ()	Creates a new element and appends it to its parent
bool Exists	Returns true if at least one element exists
int Count	Returns the count of elements
void Remove ()	Deletes all occurrences of the element from its parent
void RemoveAt (int index)	Deletes the occurrence of the element specified by the index
System.Collections.IEnumerator GetEnumerator ()	Returns an object for iterating instances of the member element.
<i>MemberType</i> Value	Sets or gets the element content (only generated if element can have mixed or simple content)
int EnumerationValue	Generated for enumeration types only. Sets or gets the element value using one of the constants generated for the possible values, reading returns Invalid if the value does not match any of the enumerated values in the schema.
Altova.Xml.Meta.Element Info	Returns an object for querying schema information (see below)

Creating XML Documents

The generated code library makes it very easy to create XML files. Let's start with an example XML file for the [example schema](#):

```
<?xml version="1.0" encoding="UTF-8"?>
<Library xsi:schemaLocation="http://www.nanonull.com/LibrarySample Library.xsd" xmlns="
http://www.nanonull.com/LibrarySample" xmlns:xsi="
http://www.w3.org/2001/XMLSchema-instance">
  <Book ISBN="0764549642" Variant="Paperback">
    <Title>The XML Spy Handbook</Title>
    <Author>Altova</Author>
  </Book>
</Library>
```

The following C# code was used to create this file. The classes Library2, LibraryType and BookType are generated out of the [schema](#) and represent the complete document, the type of

the <Library> element, and the complex type BookType, which is the type of the <Book> element. Note that the automatic name de-collision renamed the Library class to Library2 to avoid a possible conflict with the library namespace name.

```
using Library;
...
protected static void Example()
{
    // create a new, empty XML document
    Library2 libDoc = Library2.CreateDocument();

    // create the root element <Library> and add it to the document
    LibraryType lib = libDoc.Library3.Append();

    // create a new <Book> and add it to the library
    BookType book = lib.Book.Append();

    // set the ISBN attribute of the book
    book.ISBN.Value = "0764549642";

    // set the Variant attribute of the book using an enumeration
    constant
    book.Variant.EnumerationValue =
    BookVariant.EnumValues.ePaperback;

    // add the <Title> and <Author> elements, and set values
    book.Title.Append().Value = "The XML Spy Handbook";
    book.Author.Append().Value = "Altova";

    // set the schema location (this is optional)
    libDoc.SetSchemaLocation("Library.xsd");

    // save the XML document to a file with default encoding
    (UTF-8). "true" causes the file to be pretty-printed.
    libDoc.SaveToFile("Library1.xml", true);
}
```

Note that all elements are created by calling Append(), and that attributes (like ISBN in the example) can simply be accessed like structure members.

Reading XML Documents

The following example reads the file we just created and demonstrates the various methods for reading XML files:

```
using Library;
...
protected static void Example()
{
    // load XML document
    Library2 libDoc = Library2.LoadFromFile("Library1.xml");

    // get the first (and only) root element <Library>
    LibraryType lib = libDoc.Library3.First;

    // it is possible to check whether an element exists:
    if (!lib.Book.Exists)
    {
        Console.WriteLine("This library is empty.");
        return;
    }

    // iteration: for each <Book>...
    foreach (BookType book in lib.Book)
```

```

    {
        // output values of ISBN attribute and (first and only)
        title element
        Console.WriteLine("ISBN: " + book.ISBN.Value);
        Console.WriteLine("Title: " + book.Title.First.Value);

        // read and compare an enumeration value
        if (book.Variant.EnumerationValue ==
BookVariant.EnumValues.ePaperback)
            Console.WriteLine("This is a paperback book.");

        // for each <Author>...
        foreach (xs.stringType author in book.Author)
            Console.WriteLine("Author: " + author.Value);

        // alternative: use count and index
        for (int j = 0; j < book.Author.Count; ++j)
            Console.WriteLine("Author: " + book.Author[j
].Value);
    }
}

```

Note the type of the book.Author member: xs.stringType is a generated class for any element with simple string content. Similar classes are generated for all other XML Schema types.

Of course you can also load a document, modify it by adding or deleting elements and attributes, and then save the changed file back to disk.

Error Handling

Errors are reported by exceptions. The following exception classes are defined in the namespace Altova:

Class	Base Class	Description
ConversionException	Exception	Exception thrown when a type conversion fails
StringParseException	ConversionException	A value in the lexical space cannot be converted to value space.
DataSourceUnavailableException	System.Exception	A problem occurred while loading an XML instance.
DataTargetUnavailableException	System.Exception	A problem occurred while saving an XML instance.

In addition, the following .NET exceptions are commonly used:

Class	Description
System.Exception	Base class for runtime errors
System.ArgumentException	A method was called with invalid argument values, or a type conversion failed.
System.FormatException	A value in the lexical space cannot be converted to value space.
System.InvalidCastException	A value cannot be converted to another type.
System.OverflowException	A source value cannot be represented in target domain.

Accessing Metadata

The generated library allows accessing static schema information via the following classes. The properties that return one of the metadata classes return null if the respective property does not exist.

Altova.Xml.Meta.SimpleType

Method/Property	Purpose
SimpleType BaseType	Returns the base type of this type
string LocalName	Returns the local name of the type
string NamespaceURI	Returns the namespace URI of the type
XmlQualifiedName QualifiedName	Returns the qualified name of this type
int MinLength	Returns the value of this facet
int MaxLength	Returns the value of this facet
int Length	Returns the value of this facet
int TotalDigits	Returns the value of this facet
int FractionDigits	Returns the value of this facet
string MinInclusive	Returns the value of this facet
string MinExclusive	Returns the value of this facet
string MaxInclusive	Returns the value of this facet
string MaxExclusive	Returns the value of this facet
string[] Enumerations	Returns a list of all enumeration facets
string[] Patterns	Returns a list of all pattern facets
WhitespaceType Whitespace	Returns the value of the whitespace facet, which is one of: Unknown Preserve Replace Collapse

Altova.Xml.Meta.ComplexType

Method/Property	Purpose
Attribute[] Attributes	Returns a list of all attributes
Element[] Elements	Returns a list of all elements
ComplexType BaseType	Returns the base type of this type
string LocalName	Returns the local name of the type
string NamespaceURI	Returns the namespace URI of the type
XmlQualifiedName QualifiedName	Returns the qualified name of this type
SimpleType ContentType	Returns the simple type of the content
Element FindElement (string localName, string namespaceURI)	Finds the element with the specified local name and namespace URI
Attribute FindAttribute (string localName, string namespaceURI)	Finds the attribute with the specified local name and namespace URI

Altova.Xml.Meta.Element

Method/Property	Purpose
ComplexType DataType	Returns the type of the element. Note that this is always a complex type even if declared as simple in the original schema. Use the ContentType property of the returned object to get the simple content type.
int MinOccurs	Returns the minOccurs value defined in the schema
int MaxOccurs	Returns the maxOccurs value defined in the schema
string LocalName	Returns the local name of the element
string NamespaceURI	Returns the namespace URI of the element
XmlQualifiedName QualifiedName	Returns the qualified name of the element

Altova.Xml.Meta.Attribute

Method/Property	Purpose
SimpleType DataType	Returns the type of the attribute content
bool Required()	Returns true if the attribute is required
string LocalName	Returns the local name of the attribute
string NamespaceURI	Returns the namespace URI of the attribute
XmlQualifiedName QualifiedName	Returns the qualified name of the attribute

20.4.5 Using generated code compatible to old versions**Compatibility Mode**

The code generator has been improved multiple times in the recent releases of Altova products, sometimes in ways that require changes to the code that uses the generated libraries. To preserve compatibility with existing code using libraries generated with earlier releases of XMLSpy, you can set the compatibility mode to the desired release. Default is version 2007r3, which creates the latest version of the code libraries described in the previous chapters.

The following chapters describe usage of the generated classes when using one of the following compatibility options:

- Code compatible to version 2005r3
- Code compatible to version 2006 to 2007

Please note that these compatibility options will be removed in a future version.

Version 2005r3:

Code generated code for XML schemas up till V2005R3 uses a static DOM document instance as parent for all nodes. This allows creating free-standing nodes that can be added to a document later, but may rise issues with multi-threading or memory management.

```
{
    LibraryType lb=new LibraryType();
    BookType book = new BookType();
    book.addISBN( new SchemaString( "0764549642" ) );
    book.addTitle( new SchemaString( "The XMLSpy Handbook" ) );
    book.addAuthor( new SchemaString( "Altova" ) );
    lb.addBook( book );

    LibraryDoc doc=new LibraryDoc();
    doc.setRootElementName( "http://www.nanonul.com/LibrarySample"; "Library" );
    doc.setSchemaLocation( "Library.xsd" );//optional
    doc.save( "Library.xml", lb );
}
```

Version 2006 to 2007:

In C# and Java code generated with version 2006 to 2007, the standard (non-parameter) constructor is not used, because it does not establish a reference to a DOM document. A new document is created first which can then be referenced by the root node. All new child nodes must then also be created in the correct document. This is accomplished by using the generated factory functions called newXXX, please see the example code below.

```
{
    LibraryDoc doc=new LibraryDoc();
    doc.setSchemaLocation( "Library.xsd" );//optional
```

```

    // create root element (with no namespace prefix)
    LibraryType lib = new LibraryType( doc, "
http://www.nanonull.com/LibrarySample", "", "Library" );

    BookType book = lib.newBookType();    // factory functions are
    generated for all members of a complex type
    book.addISBN( new SchemaString( "0764549642" ) );
    book.addTitle( new SchemaString( "The XML Spy Handbook" ) );
    book.addAuthor( new SchemaString( "Altova" ) );
    lib.addBook( book );

    doc.save( "Library1.xml", lib );
}

```

Version 2007r3:

The code generated by version 2007r3 has been completely redesigned to allow for multi-threading, avoid name collisions, and simplify the usage of the libraries. For details, see [Using the generated code library](#).

Creating XML files (XMLSpy 2006)

Please note: This topic describes code generation when using the option **Generate code compatible to V2006-2007** in the code generation dialog.

Java:

To give a general overview, here is the code to create a test file:

```

protected static void example() throws Exception {
    LibraryDoc doc = new LibraryDoc();
    doc.setSchemaLocation("Library.xsd"); // optional
    LibraryType lib = new LibraryType(doc,
"http://www.nanonull.com/LibrarySample", "", "Library");
    BookType book = lib.newBook();
    book.addISBN( new SchemaString( "0764549642" ) );
    book.addTitle( new SchemaString( "The XML Spy Handbook" ) );
    book.addAuthor( new SchemaString( "Altova" ) );
    lib.addBook( book );
    doc.save("Library1.xml", lib);
}

```

The idea of the generated classes is the following:

Create a document and associate it with the root element of the generated schema so that the tree is stored in the document.

First we create a new document:

```
LibraryDoc doc = new LibraryDoc();
```

Optionally we can also set the schema location.

```
doc.setSchemaLocation( "Library.xsd" );
```

We create the root element, passing to the method the document object that we have already generated, the namespace of `library.xsd`, and the name of the root element in the schema:

```
LibraryType lib = new LibraryType(doc,
```



```
"http://www.nanonull.com/LibrarySample", "", "Library" );
```

LibraryType is a class that was generated. It is similar to the `<xs:element name="Library">` in the schema file. This is the root element for the xml files being generated (as in generate sample xml file).

We now have a new Library object. All we have to do is fill it with Book objects. Look at the schema again and notice that books are stored as a sequence within the Library.

The next step is to make an object of **BookType** and fill the new object.

```
BookType book = lib.newBook();
```

Definition of BookType in the schema file:

```
<xs:complexType name="BookType">
  <xs:sequence>
    <xs:element name="Title" type="xs:string"/>
    <xs:element name="Author" type="xs:string" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="ISBN" type="xs:string" use="required"/>
</xs:complexType>
```

We will insert an ISBN number, a title and an author.

```
book.addISBN( new SchemaString( "0764549642" ) );
book.addTitle( new SchemaString( "The XML Spy Handbook" ) );
book.addAuthor( new SchemaString( "Altova" ) );
```

We filled the BookType object, and we now have to add it to the library sequence:

```
lib.addBook(book);
```

Finally, save the file. The first parameter is the file name, the second is the root element itself.

```
doc.save( "Library1.xml", lib );
```

C++:

The C++ implementation is very similar to the Java one, we will therefore only discuss the differences between the two.

The function to be inserted above the main function is as follows:

```
void Example()
{
    CLibraryDoc doc;
    CLibraryType lib;
    doc.SetRootElementName(_T("http://www.nanonull.com/LibrarySample"), _T(
"Library"));
    doc.SetSchemaLocation(_T("Library.xsd")); // optional
    CBookType book;
    book.AddISBN( _T("0764549642") );
    book.AddTitle( _T("The XML Spy Handbook") );
    book.AddAuthor( _T("Altova") );
    lib.AddBook( book );
    doc.Save(_T("Library1.xml"), lib);
```

```
}
```

First off, you'll notice that all classes begin with a capital C - the rest of the class naming stays the same. All member function names start with a capital letter.

Please take care with the `_T` macros. The macro ensures that the string constant is stored correctly, whether you're compiling for Unicode or non-Unicode programs.

C#:

```
protected static void Example()
{
    LibraryDoc doc = new LibraryDoc();
    doc.SetSchemaLocation( "Library.xsd" ); // optional
    // create root element with no namespace prefix
    LibraryType lib = new LibraryType(doc, "
http://www.nanonull.com/LibrarySample", "", "Library" );
    BookType book = lib.NewBook(); // factory functions are generated for all
members of a complex type
    book.AddISBN( new SchemaString( "0764549642" ) );
    book.AddTitle( new SchemaString( "The XML Spy Handbook" ) );
    book.AddAuthor( new SchemaString( "Altova" ) );
    lib.AddBook( book );
    doc.Save( "Library1.xml", lib );
}
```

Creating XML files (XMLSpy 2005)

Please note: This topic describes code generation when using the option **Generate code compatible to V2005R3** in the code generation dialog.

Java:

To give a general overview, here is the code to create a test file:

```
protected static void example() {
    LibraryType lib = new LibraryType();
    BookType book = new BookType();
    book.addISBN( new SchemaString( "0764549642" ) );
    book.addTitle( new SchemaString( "The XML Spy Handbook" ) );
    book.addAuthor( new SchemaString( "Altova" ) );
    lib.addBook( book );

    LibraryDoc doc = new LibraryDoc();
    doc.setRootElementName( "http://www.nanonull.com/LibrarySample", "Library"
);
    doc.setSchemaLocation( "Library.xsd" ); // optional
    doc.save( "Library1.xml", lib );
}
```

The idea of the generated classes is the following:
Generate the tree you want to save in the new file, create a document and store the tree in it.

The first line:

```
LibraryType lib = new LibraryType();
```

LibraryType is a class that was generated. It is similar to the `<xs:element name="Library">` in the schema file. This is the root element for the xml files being generated (as in generate sample xml file).

We now have a new Library object. All we have to do is fill it with Book objects. Look at the schema again and notice that books are stored as a sequence within the Library.

The next step is to make an object of **BookType** and fill the new object.

```
BookType book = new BookType();
```

Definition of BookType in the schema file:

```
<xs:complexType name="BookType">
  <xs:sequence>
    <xs:element name="Title" type="xs:string"/>
    <xs:element name="Author" type="xs:string" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="ISBN" type="xs:string" use="required"/>
</xs:complexType>
```

We will insert an ISBN number, a title and an author.

```
book.addISBN( new SchemaString( "0764549642" ) );
book.addTitle( new SchemaString( "The XML Spy Handbook" ) );
book.addAuthor( new SchemaString( "Altova" ) );
```

We filled the BookType object, and we now have to add it to the sequence of **internationalPriceTypes** within Test_Empty:

```
lib.addBook(book);
```

Done, the tree is complete! All we have to do now, is create a new document and define the namespace and name of the root element:

```
LibraryDoc doc = new LibraryDoc();
doc.setRootElementName( "http://www.nanonull.com/LibrarySample", "Library"
);
```

Optionally we can also set the schema location.

```
doc.setSchemaLocation( "Library.xsd" );
```

Finally, save the file, - the first parameter is the file name, the second is the root element itself.

```
doc.save( "Library1.xml", lib );
```

C++:

The C++ implementation is very similar to the Java one, we will therefore only discuss the differences between the two.

The function to be inserted above the main function is as follows:

```
void Example()
{
    CLibraryType lib;
```

```

CBookType book;
book.AddISBN( _T("0764549642") );
book.AddTitle( _T("The XML Spy Handbook") );
book.AddAuthor( _T("Altova") );
lib.AddBook( book );

CLibraryDoc doc;
doc.SetRootElementName( _T("http://www.nanonull.com/LibrarySample"), _T(
"Library") );
doc.SetSchemaLocation( _T("Library.xsd") ); // optional
doc.Save( _T("Library1.xml"), lib );
}

```

First off, you'll notice that all classes begin with a capital C - the rest of the class naming stays the same. All member function names start with a capital letter.

Please take care with the `_T` macros. The macro ensures that the string constant, is stored correctly, whether you're compiling for Unicode or non Unicode programs.

C#:

```

protected static void Example()
{
    LibraryType lib = new LibraryType();
    BookType book = new BookType();
    book.AddISBN( new SchemaString( "0764549642" ) );
    book.AddTitle( new SchemaString( "The XML Spy Handbook" ) );
    book.AddAuthor( new SchemaString( "Altova" ) );
    lib.AddBook( book );

    LibraryDoc doc = new LibraryDoc();
    doc.SetRootElementName( "http://www.nanonull.com/LibrarySample", "Library"
);
    doc.SetSchemaLocation( "Library.xsd" ); // optional
    doc.Save( "Library1.xml", lib );
}

```

Opening and parsing existing XML files (XMLSpy 2006)

Please note: This topic describes code generation when using the option **Generate code compatible to V2006-2007** in the code generation dialog.

We will now use the **Library1.xml** file generated with our sample program. Please make sure that this file exists, or change the path to it in the load command.

Java:

Sample source code:

```

protected static void example2() {

    LibraryDoc doc = new LibraryDoc();
    LibraryType lib = new LibraryType(doc.load("Library1.xml"));

    if (!lib.hasBook()) {
        System.out.println("This library is empty");
    }
}

```

```

        return;
    }

    for (int i = 0; i < lib.getBookCount(); i++) {
        BookType book;
        try {
            book = lib.getBookAt(i);

            System.out.println("ISBN: " + book.getISBN().toString());
            System.out.println("Title: " + book.getTitle().toString());

            for (int j = 0; j < book.getAuthorCount(); j++) {
                System.out.println("Author: " +
book.getAuthorAt(j).toString());
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

in the **main()** function add a call to:

```

    example2();
after
    example();

```

The rest is reasonably straightforward.

First create a new variable of type Document, and one for the root element, which is immediately filled by the **doc.load** function. The only parameter it needs, is the path to the xml file being used.

```

LibraryDoc doc = new LibraryDoc();
LibraryType lib = new LibraryType( doc.load( "Library1.xml" ) );

```

We now have a document and know that lib contains a sequence of BookType objects.

The next step is to dump the values of all of the Book elements:

```

if( !lib.hasBook() ) {

```

Why check for objects? The call just shows that a function exists that checks this for you.

Step through the books and output their values:

```

    for( int i = 0; i < lib.getBookCount(); i++ ) {
        BookType book = lib.getBookAt( i );

        System.out.println( "ISBN: " + book.getISBN().asString() );
        System.out.println( "Title: " + book.getTitle().asString() );

        for( int j = 0; j < book.getAuthorCount(); j++ ) {
            System.out.println( "Author: " + book.getAuthorAt( j ).asString()
        );
    }
}

```

We're finished, all the values are listed in your output window.

C++:

This section only explains the differences to the java code.

```
void Example2()
{
    CLibraryDoc doc;
    CLibraryType lib = doc.Load( _T("Library1.xml") );

    if ( !lib.HasBook() )
    {
        cout << "This library is empty" << endl;
        return;
    }

    for ( int i = 0; i < lib.GetBookCount(); i++ )
    {
        CBookType book = lib.GetBookAt( i );
        cout << "ISBN: " << book.GetISBN() << endl;
        cout << "Title: " << book.GetTitle() << endl;

        for ( int j = 0; j < book.GetAuthorCount(); j++ )
        {
            cout << "Author: " << book.GetAuthorAt( j ) << endl;
        }
    }
}
```

All member functions start with a capital letter, and all class names start with a C.

C#:

This section only explains the differences to the java code.

```
protected static void Example2()
{
    LibraryDoc doc = new LibraryDoc();
    LibraryType lib = new LibraryType( doc.Load( "Library1.xml" ) );

    if ( !lib.HasBook() )
    {
        Console.WriteLine( "This library is empty" );
        return;
    }

    for ( int i = 0; i < lib.GetBookCount(); i++ )
    {
        BookType book = lib.GetBookAt( i );
        Console.WriteLine( "ISBN: {0}", book.GetISBN() );
        Console.WriteLine( "Title: {0}", book.GetTitle() );

        for( int j = 0; j < book.GetAuthorCount(); j++ )
        {
            Console.WriteLine( "Author: {0}", book.GetAuthorAt( j ) );
        }
    }
}
```

```

    }
}

```

All member functions start with a capital letter.

Opening and parsing existing XML files (XMLSpy 2005)

Please note: This topic describes code generation when using the option **Generate code compatible to V2005R3** in the code generation dialog.

We will now use the **Library1.xml** file generated with our sample program. Please make sure that this file exists, or change the path to it in the load command.

Java:

Sample source code:

```

protected static void example2() {
    LibraryDoc doc = new LibraryDoc();
    LibraryType lib = new LibraryType( doc.load( "Library1.xml" ) );

    if( !lib.hasBook() ) {
        System.out.println( "This library is empty" );
        return;
    }

    for( int i = 0; i < lib.getBookCount(); i++ ) {
        BookType book = lib.getBookAt( i );
        System.out.println( "ISBN: " + book.getISBN().asString() );
        System.out.println( "Title: " + book.getTitle().asString() );

        for( int j = 0; j < book.getAuthorCount(); j++ ) {
            System.out.println( "Author: " + book.getAuthorAt( j
        ).asString());
        }
    }
}

```

in the **main()** function add a call to:

```

    example2();
after
    example();

```

The rest is reasonably straightforward.

First create a new variable of type Document, and one for the root element, which is immediately filled by the **doc.load** function. The only parameter it needs, is the path to the xml file being used.

```

LibraryDoc doc = new LibraryDoc();
LibraryType lib = new LibraryType( doc.load( "Library1.xml" ) );

```

We now have a document and know that lib contains a sequence of BookType objects.

The next step is to dump the values of all of the Book elements:

```
if( !lib.hasBook() ) {
```

Why check for objects? The call just shows that a function exists, that checks this for you.

Step through the books and dump their values:

```
for( int i = 0; i < lib.getBookCount(); i++ ) {
    BookType book = lib.getBookAt( i );

    System.out.println( "ISBN: " + book.getISBN().asString() );
    System.out.println( "Title: " + book.getTitle().asString() );

    for( int j = 0; j < book.getAuthorCount(); j++ ) {
        System.out.println( "Author: " + book.getAuthorAt( j ).asString()
    );
    }
}
```

We're finished, all the values are listed in your output window.

C++:

This section only explains the differences to the java code.

```
void Example2()
{
    CLibraryDoc doc;
    CLibraryType lib = doc.Load( _T("Library1.xml") );

    if( !lib.HasBook() )
    {
        cout << "This library is empty" << endl;
        return;
    }

    for( int i = 0; i < lib.GetBookCount(); i++ )
    {
        CBookType book = lib.GetBookAt( i );

        cout << "ISBN: " << book.GetISBN() << endl;
        cout << "Title: " << book.GetTitle() << endl;

        for( int j = 0; j < book.GetAuthorCount(); j++ )
        {
            cout << "Author: " << book.GetAuthorAt( j ) << endl;
        }
    }
}
```

All member functions start with a capital letter, and all Classnames start with a C.

C#:

This section only explains the differences to the java code.

```
protected static void Example2()
```



```
{
    LibraryDoc doc = new LibraryDoc();
    LibraryType lib = new LibraryType( doc.Load( "Library1.xml" ) );

    if( !lib.HasBook() )
    {
        Console.WriteLine( "This library is empty" );
        return;
    }

    for( int i = 0; i < lib.GetBookCount(); i++ )
    {
        BookType book = lib.GetBookAt( i );

        Console.WriteLine( "ISBN: {0}", book.GetISBN() );
        Console.WriteLine( "Title: {0}", book.GetTitle() );

        for( int j = 0; j < book.GetAuthorCount(); j++ )
        {
            Console.WriteLine( "Author: {0}", book.GetAuthorAt( j ) );
        }
    }
}
```

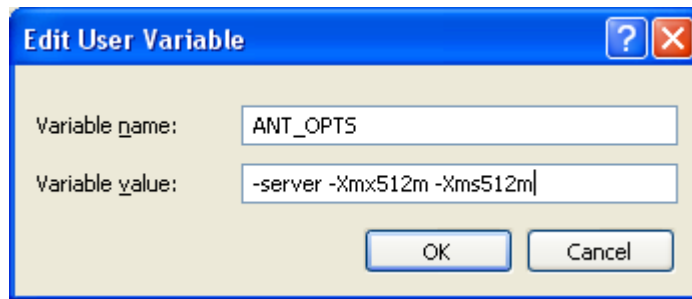
All member functions start with a capital letter.

20.5 Code generation tips

Out-of-memory exceptions during Java compilation and how to resolve them

Complex schemas can produce a large amount of code, which might cause a `java.lang.OutOfMemory` exception during compilation using Ant. To rectify this:

- Add the environment variable **ANT_OPTS**, which sets specific Ant options such as the memory to be allocated to the compiler, and add values as shown below.



- To make sure that the compiler and the generated code run in the same process as Ant, change the **fork** attribute, in `build.xml`, to **false**.

You may need to customize the values depending on the amount of memory in your machine and the size of the project you are working with. For more details please see your Java VM documentation.

Reserving method names

When customizing code generation using the supplied `spl` files, it might be necessary to reserve names to avoid collisions with other symbols. To do this:

1. Navigate to subdirectory corresponding to the programming language of the **spl** subdirectory of the program installation directory e.g. `C:\Program Files\Altova\XMLSpy2012\spl\java\`.
2. Open the **settings.spl** file and insert a new line into the reserve section e.g. `reserve "myReservedWord"`.
3. Regenerate the program code.

XML Schema support

The following XML Schema constructs are translated into code:

- XML Namespaces: Mapped to namespaces in the target programming language

Simple types

- Built-in XML Schema types: Mapped to native primitive types or classes in the Altova types library
- Derived by extension
- Derived by restriction
- Facets
- Enumerations
- Patterns

Complex types

- Built-in anyType node
- User-defined complex types
- Derived by extension: Mapped to derived classes
- Derived by restriction
- Complex content
- Simple content
- Mixed content

20.6 Code generator options

The menu option **DTD/Schema | Generate Program Code** lets you specify the target programming language as well as specific code generation settings.

C++ and C# Settings tab:

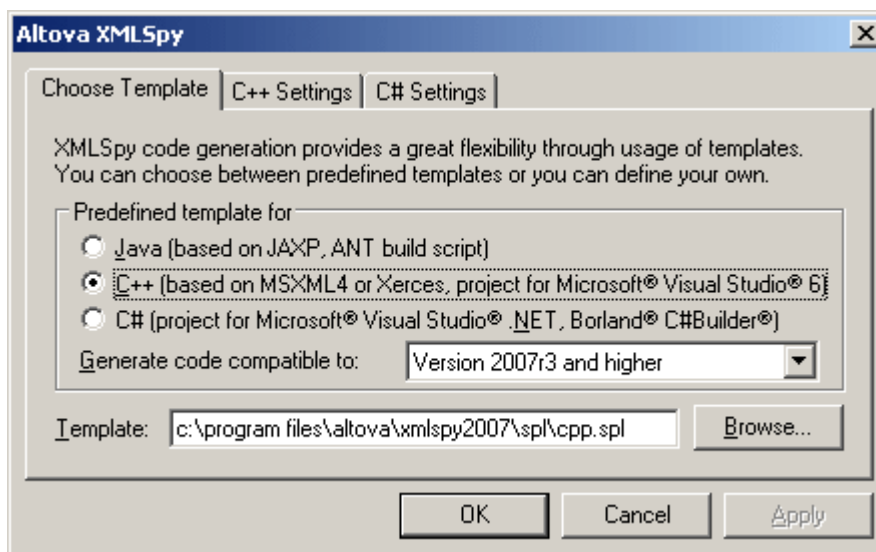
C++ Settings

- Microsoft Visual Studio 2010, 2008, 2005, Visual C++ 6.0 project file, or makefile for Linux with GCC compiler.
- generate code using either MSXML 6.0 or Apache Xerces XML library
- generate static libraries, or Dynamic-link libraries
- generate code with or without MFC support

C# Settings

Select the type of project file you want to generate:

- Microsoft Visual Studio 2010, 2008, 2005, project file
- Mono Makefile support was removed in MapForce version 2011. Mono users can use Visual Studio solution files with Mono's xbuild.



Compatibility Mode:

The code generator has been improved multiple times in the recent releases of Altova products, sometimes in ways that require changes to the code that uses the generated libraries. To preserve compatibility with existing code using libraries generated with earlier releases of XMLSpy, you can set the compatibility mode to the desired release. Default is version 2007r3, which creates the latest version of the code libraries.

- Please make it a point to use the new code generation features i.e., set the version to the latest available.
- The availability of the Compatibility Mode is only temporary, it will be removed in a future version.

20.7 The way to SPL (Spy Programming Language)

This section gives an overview of Spy Programming Language, the code generator's template language.

It is assumed that you have prior programming experience, and are familiar with operators, functions, variables and classes, as well as the basics of object-oriented programming - which is used heavily in SPL. You should also have detailed knowledge of XML Schema.

The templates used by XMLSpy are supplied in the ...\\XMLSpy2012\\spl folder. You can use these files as an aid to help you in developing your own templates.

How code generator works

Inputs to the code generator are the template files (.spl) and the object model provided by XMLSpy. The template files contain SPL instructions for creating files, reading information from the object model and performing calculations, interspersed with literal code fragments in the target programming language.

The template file is interpreted by the code generator and outputs .cpp, .java, .cs source code files, project files, or any other type of file depending on the template. The source code can then be compiled into an executable file that accesses XML data described by the schema file.

SPL files have access to a wide variety of information that is collated from the source schemas. Please note that an SPL file is not tied to a specific schema, but allows access to all schemas! Make sure you write your SPL files generically, avoid structures etc. which apply to specific schemas!

Example: Creating a new file in SPL:

```
[ create "test.cpp"]
#include "stdafx.h"
[ close]
```

This is a very basic SPL file. It creates a file named **test.cpp**, and places the include statement within it. The close command completes the template.

20.7.1 Basic SPL structure

An SPL file contains literal text to output, interspersed with code generator instructions.

Code generator instructions are enclosed in square brackets '[' and ']'.

Multiple statements can be included in a bracket pair. Additional statements have to be separated by a new line or a colon ':'.

Valid examples are:

```
[ $x = 42
  $x = $x + 1 ]
```

or

```
[ $x = 42: $x = $x + 1 ]
```

Adding text to files

Text not enclosed by [and], is written directly to the current output file. If there is no current output file, the text is ignored (see [Using files](#) how to create an output file).

To output literal square brackets, escape them with a backslash: \[and \]; to output a backslash use \\.

Comments

Comments inside an instruction block always begin with a ' character, and terminate on the next line, or at a block close character].

20.7.2 Declarations

The following statements are evaluated while parsing the SPL template file. They are **not** affected by flow control statements like conditions, loops or subroutines, and are always evaluated exactly once.

These keywords, like all keywords in SPL, are not case sensitive.

Remember that all of these declarations must be inside a block delimited by square brackets.

map *mapname key to value* [, *key to value*]...

This statement adds information to a map. See below for specific uses.

map schemanativetype *schematype to typespec*

The specified built-in XML Schema type will be mapped to the specified native type or class, using the specified formatter. This setting applies only to code generation for version 2007r3 and higher. Typespec is a native type or class name, followed by a comma, followed by the formatter class instance.

Example:

```
map schemanativetype "double" to "double,Altova::DoubleFormatter"
```

map type *schematype to classname*

The specified built-in XML Schema type will be mapped to the specified class. This setting applies only to code generation for version 2007 or lower.

Example:

```
map type "float" to "CSchemaFloat"
```

default setting is value

This statement allows you to affect how class and member names are derived from the XML Schema.

Note that the setting names are case sensitive.

Example:

```
default "InvalidCharReplacement" is "_"
```

Setting name	Explanation
ValidFirstCharSet	Allowed characters for starting an identifier
ValidCharSet	Allowed characters for other characters in an identifier
InvalidCharReplacement	The character that will replace all characters in names that are not in the ValidCharSet
AnonTypePrefix	Prefix for names of anonymous types*
AnonTypeSuffix	Suffix for names of anonymous types*
ClassNamePrefix	Prefix for generated class names
ClassNameSuffix	Suffix for generated class names
EnumerationPrefix	Prefix for symbolic constants declared for enumeration values

EnumerationUpper"on" to convert the enumeration constant names to upper case
Case
FallbackName If a name consists only of characters that are not in ValidCharSet, use this one

* Names of anonymous types are built from AnonTypePrefix + element name + AnonTypeSuffix

reserve word

Adds the specified word to the list of reserved words. This ensures that it will never be generated as a class or member name.

Example:

```
reserve "while"
```

include filename

Example:

```
include "Module.cpp"
```

includes the specified file as SPL source. This allows you to split your template into multiple files for easier editing and handling.

20.7.3 Variables

Any non-trivial SPL file will require variables. Some variables are [predefined](#) by the code generator, and new variables may be created simply by assigning values to them.

The **\$** character is used when **declaring** or **using** a variable, a variable name is always prefixed by **\$**.

Variable names are **case sensitive**.

Variables types:

- integer - also used as boolean, where 0 is false and everything else is true
- string
- object - provided by XMLSpy
- iterator - see [foreach](#) statement

Variable types are declared by first assignment:

```
[ $x = 0 ]
```

x is now an integer.

```
[ $x = "teststring" ]
```

x is now treated as a string.

Strings

String constants are always enclosed in double quotes, like in the example above. **\n** and **\t** inside double quotes are interpreted as newline and tab, **\"** is a literal double quote, and **** is a backslash. String constants can also span multiple lines.

String concatenation uses the **&** character:

```
[ $BasePath = $outputpath & "/" & $JavaPackageDir ]
```

Objects

Objects represent the information contained in the XML schema. Objects have **properties**, which can be accessed using the `.` operator. It is not possible to create new objects in SPL (they are predefined by the code generator, derived from the input schema), but it is possible to assign objects to variables.

Example:

```
class [= $class.Name]
```

This example outputs the word "class", followed by a space and the value of the **Name** property of the **\$class** object.

20.7.4 Predefined variables

After a Schema file is analyzed by the code generator, the objects in the table below exist in the Template Engine.

Current object model (for code compatibility to version 2007r3 and later)

Name	Type	Description
\$schematype	integer	1 for DTD, 2 for XML Schema
\$TheLibrary	Library	The library derived from the XML Schema or DTD
\$module	string	Name of the source Schema without extension
\$outputpath	string	The output path specified by the user, or the default output path

Old object model (for code compatibility up to version 2007)

Name	Type	Description
\$schematype	integer	1 for DTD, 2 for XML Schema
\$namespaces	Namespace collection	Collection of Namespace objects
\$classes	Class collection	All the complex types, elements,... in a flat view
\$module	string	name of the source Schema without extension
\$outputpath	string	The output path specified by the user, or the default output path

For **C++** generation only:

Name	Type	Description
\$domtype	integer	1 for MSXML, 2 for Xerces
\$XercesVersion	integer	2 for Xerces 2.x, 3 for Xerces 3.x
\$libtype	integer	1 for static LIB, 2 for DLL
\$mfc	boolean	True if MFC support is enabled
\$vc6project	boolean	True if Visual C++ 6.0 project files are to be generated
\$VS2005Project	boolean	True if Visual C++ 2005 project files are to be generated
\$VS2008Project	boolean	True if Visual C++ 2008 project files are to be generated
\$VS2010Project	boolean	True if Visual C++ 2010 project files are to be generated

For **C#** generation only:

Name	Type	Description

\$CreateVS2005Project	boolean	True if Visual Studio 2005 project files are to be generated
\$CreateVS2008Project	boolean	True if Visual Studio 2008 project files are to be generated
\$CreateVS2010Project	boolean	True if Visual Studio 2010 project files are to be generated

20.7.5 Creating output files

These statements are used to create output files from the code generation.

Remember that all of these statements must be inside a block delimited by square brackets.

create filename

creates a new file. The file has to be closed with the **close** statement. All following output is written to the specified file.

Example:

```
[ create $outputpath & "/" & $JavaPackageDir & "/" & $application.Name &
".java"
package [ =$JavaPackageName];

public class [ =$application.Name] Application {
...
}
[ close]
```

close

closes the current output file.

= \$variable

writes the value of the specified variable to the current output file.

Example:

```
[ $x = 20+3]
The result of your calculation is [ =$x] - so have a nice day!
```

- The file output will be:

```
The result of your calculation is 23 - so have a nice day!
```

write string

writes the string to the current output file.

Example:

```
[ write "C" & $name]
```

This can also be written as:

```
C[ =$name]
```

filecopy source to target

copies the source file to the target file, without any interpretation.

Example:

```
filecopy "java/mapforce/mapforce.png" to $outputpath & "/" & $JavaPackageDir &
"/mapforce.png"
```

20.7.6 Operators

Operators in SPL work like in most other programming languages.

List of SPL operators in descending precedence order:

.	Access object property
()	Expression grouping
true	boolean constant "true"
false	boolean constant "false"
&	String concatenation
-	Sign for negative number
not	Logical negation
*	Multiply
/	Divide
%	Modulo
+	Add
-	Subtract
<=	Less than or equal
<	Less than
>=	Greater than or equal
>	Greater than
=	Equal
<>	Not equal
and	Logical conjunction (with short circuit evaluation)
or	Logical disjunction (with short circuit evaluation)
=	Assignment

20.7.7 Conditions

SPL allows you to use standard "if" statements. The syntax is as follows:

```
if condition
    statements
else
    statements
endif
```

or, without else:

```
if condition
    statements
endif
```

Please note that there are no round brackets enclosing the condition!

As in any other programming language, conditions are constructed with logical and comparison [operators](#).

Example:

```
[ if $namespace.ContainsPublicClasses and $namespace.Prefix <> ""  
  whatever you want ['inserts whatever you want, in the resulting file]  
[ endif]
```

Switch

SPL also contains a multiple choice statement.

Syntax:

```
switch $variable  
  case X:  
    statements  
  case Y:  
  case Z:  
    statements  
  default:  
    statements  
endswitch
```

The case labels must be constants or variables.

The switch statement in SPL does not fall through the cases (as in C), so there is no need for a "break" statement.

20.7.8 Collections and foreach

Collections and iterators

A collection contains multiple objects - like a ordinary array. Iterators solve the problem of storing and incrementing array indexes when accessing objects.

Syntax:

```
foreach iterator in collection  
  statements  
next
```

Example:

```
[ foreach $class in $classes  
  if not $class.IsInternal  
    ] class [=$class.Name];  
[ endif  
next]
```

Example 2:

```
[ foreach $i in 1 To 3  
  Write "// Step " & $i & "\n"  
  \ Do some work  
next]
```

The first line:

\$classes is the [global object](#) of all generated types. It is a collection of single class objects.

Foreach steps through all the items in \$classes, and executes the code following the instruction, up to the **next** statement, for each of them.

In each iteration, **\$class** is assigned to the next class object. You simply work with the class

object instead of using, `classes[i]->Name()`, as you would in C++.

All collection iterators have the following additional properties:

Index	The current index, starting with 0
IsFirst	true if the current object is the first of the collection (index is 0)
IsLast	true if the current object is the last of the collection
Current	The current object (this is implicit if not specified and can be left out)

Example:

```
[foreach $enum in $facet.Enumeration
  if not $enum.IsFirst
  ], [
  endif
] "[=$enum.Value]"[
next]
```

Collection manipulation routines:

collection **SortByName**(bAscending)

returns a collection whose elements are sorted by name (case sensitive) in ascending or descending order.

collection **SortByNameNoCase**(bAscending)

returns a collection whose elements are sorted by name (case insensitive) in ascending or descending order

Example:

```
$SortedNestedClassifier = $Class.nestedClassifier.SortByNameNoCase( true )
```

collection **SortByKind**(bAscending)

returns a collection whose elements are sorted by kind names (e.g. "Class", "Interface",...) in ascending or descending order.

collection **SortByKindAndName**(bAscendingKind, bAscendingName)

returns a collection whose elements are sorted by kind (e.g. "Class", "Interface",...) in ascending or descending order and if the kinds are equal by name (case sensitive in ascending or descending order)

collection **SortByKindAndNameNoCase**(bAscending)

returns a collection whose elements are sorted by kind (e.g. "Class", "Interface",...) in ascending or descending order and if the kinds are equal by name (case insensitive in ascending or descending order)

20.7.9 Subroutines

Code generator supports subroutines in the form of procedures or functions.

Features:

- By-value and by-reference passing of values
- Local/global parameters (local within subroutines)
- Local variables
- Recursive invocation (subroutines may call themselves)

Subroutine declaration

Subroutines

Syntax example:

```
Sub SimpleSub()  
... lines of code  
EndSub
```

- **Sub** is the keyword that denotes the procedure.
- **SimpleSub** is the name assigned to the subroutine.
- Round **parenthesis** can contain a parameter list.
- The code block of a subroutine starts immediately after the closing parameter parenthesis.
- **EndSub** denotes the end of the code block.

Please note:

Recursive or cascaded subroutine **declaration** is not permitted, i.e. a subroutine may not contain another subroutine.

Parameters

Parameters can also be passed by procedures using the following syntax:

- All parameters must be variables
- Variables must be prefixed by the **\$** character
- Local variables are defined in a subroutine
- Global variables are declared explicitly, outside of subroutines
- Multiple parameters are separated by the comma character **,** within round parentheses
- Parameters can pass values

Parameters - passing values

Parameters can be passed in two ways, by value and by reference, using the keywords **ByVal** and **ByRef** respectively.

Syntax:

```
' define sub CompleteSub()  
[ Sub CompleteSub( $param, ByVal $paramByValue, ByRef $paramByRef )  
] ...
```

- **ByVal** specifies that the parameter is passed by value. Note that most objects can only be passed by reference.
- **ByRef** specifies that the parameter is passed by reference. This is the default if neither **ByVal** nor **ByRef** is specified.

Function return values

To return a value from a subroutine, use the **return** statement. Such a function can be called from within an expression.

Example:

```
' define a function  
[ Sub MakeQualifiedName( ByVal $namespacePrefix, ByVal $localName )  
if $namespacePrefix = ""
```

```
    return $localName
else
    return $namespacePrefix & ":" & $localName
endif
EndSub
]
```

Subroutine invocation

Use **call** to invoke a subroutine, followed by the procedure name and parameters, if any.

```
    Call SimpleSub( )
or,
    Call CompleteSub( "FirstParameter", $ParamByValue, $ParamByRef )
```

Function invocation

To invoke a function (any subroutine that contains a **return** statement), simply use its name inside an expression. Do not use the **call** statement to call functions.

Example:

```
    $QName = MakeQualifiedName( $namespace, "entry" )
```

Subroutine example

Highlighted example showing subroutine declaration and invocation.

```
A sample SPL file:
[create $outputpath & $module & "output.txt"

' define sub SimpleSub()
sub SimpleSub()
]SimpleSub() called
[endsub

' execute sub SimpleSub()
call SimpleSub()

$ParamByValue = "Original Value"
]ParamByValue = [= $ParamByValue]
[$ParamByRef = "Original Value"
]ParamByRef = [= $ParamByRef]

' define sub CompleteSub()
[sub CompleteSub( $param, byval $paramByValue, byref $paramByRef )
]CompleteSub called.
    param = [= $param]
    paramByValue = [= $paramByValue]
    paramByRef = [= $paramByRef]
    [$ParamByRef = "Local Variable"
    $paramByValue = "new value"
    $paramByRef = "new value"
    ] Set values inside sub
    [$ParamByRef = "Local Variable"
    $paramByValue = "new value"
    $paramByRef = "new value"
    ]CompleteSub finished.
[endsub

' run sub CompleteSub()
call CompleteSub( "FirstParameter", $ParamByValue, $ParamByRef )
]
ParamByValue=[= $ParamByValue]
ParamByRef=[= $ParamByRef]
[
close
]
```

The same sample code:

```
[create $outputpath & $module & "output.txt"

' define sub SimpleSub()
Sub SimpleSub()
]SimpleSub() called
[endsub

' execute sub SimpleSub()
Call SimpleSub()

$ParamByValue      = "Original Value"
]ParamByValue      = [= $ParamByValue]
[$ParamByRef = "Original Value"
]ParamByRef      = [= $ParamByRef]

' define sub CompleteSub()

```

```

[ Sub CompleteSub( $param, ByVal $paramByValue, ByRef $paramByRef )
] CompleteSub called.
    param = [= $param]
    paramByValue = [= $paramByValue]
    paramByRef = [= $paramByRef]
[ $ParamByRef = "Local Variable"
$paramByValue = "new value"
$paramByRef = "new value"
]      Set values inside Sub
[ $ParamByRef = "Local Variable"
$paramByValue = "new value"
$paramByRef = "new value"
] CompleteSub finished.
[ endsub

' run sub CompleteSub()
Call CompleteSub( "FirstParameter", $ParamByValue, $ParamByRef )
]
ParamByValue=[= $ParamByValue]
ParamByRef=[= $ParamByRef]
[
Close
]

```

20.7.10 Built in Types

The section describes the properties of the built-in types used in the [predefined variables](#) which describe the parsed schema.

Library

This object represents the whole library generated from the XML Schema or DTD.

Property	Type	Description
SchemaNamespaces	Namespace collection	Namespaces in this library
SchemaFilename	string	Name of the XSD or DTD file this library is derived from
SchemaType	integer	1 for DTD, 2 for XML Schema
Guid	string	A globally unique ID
CodeName	string	Generated library name (derived from schema file name)

Namespace

One namespace object per XML Schema namespace is generated. Schema components that are not in any namespace are contained in a special namespace object with an empty NamespaceURI.

Note that for DTD, namespaces are also derived from attributes whose names begin with "xmlns".

Property	Type	Description
CodeName	string	Name for generated code (derived from prefix)
LocalName	string	Namespace prefix
NamespaceURI	string	Namespace URI
Types	Type collection	All types contained in this namespace
Library	Library	Library containing this namespace

Type

This object represents a complex or simple type. It is used to generate a class in the target language.

There is one additional type per library that represents the document, which has all possible root elements as members.

Anonymous types have an empty LocalName.

Property	Type	Description
CodeName	string	Name for generated code (derived from local name or parent declaration)
LocalName	string	Original name in the schema
Namespace	Namespace	Namespace containing this type
Attributes	Member collection	Attributes contained in this type*
Elements	Member collection	Child elements contained in this type
IsSimpleType	boolean	True for simple types, false for complex types
IsDerived	boolean	True if this type is derived from another type, which is also represented by a Type object
IsDerivedByExtension	boolean	True if this type is derived by extension
IsDerivedByRestriction	boolean	True if this type is derived by restriction
IsDerivedByUnion	boolean	True if this type is derived by union
IsDerivedByList	boolean	True if this type is derived by list
BaseType	Type	The base type of this type (if IsDerived is true)
IsDocumentRootType	boolean	True if this type represents the document itself
Library	Library	Library containing this type
IsFinal	boolean	True if declared as final in the schema
IsMixed	boolean	True if this type can have mixed content
IsAbstract	boolean	True if this type is declared as abstract
IsGlobal	boolean	True if this type is declared globally in the schema
IsAnonymous	boolean	True if this type is declared locally in an element

For simple types only:

Property	Type	Description
IsNativeBound	boolean	True if native type binding exists
NativeBinding	NativeBinding	Native binding for this type
Facets	Facets	Facets of this type
Whitespace	string	Shortcut to the Whitespace facet

* Complex types with text content (these are types with mixed content and complexType with simpleContent) have an additional unnamed attribute member that represents the text content.

Member

This object represents an attribute or element in the XML Schema. It is used to create class members of types.

Property	Type	Description

CodeName	string	Name for generated code (derived from local name or parent declaration)
LocalName	string	Original name in the schema. Empty for the special member representing text content of complex types.
NamespaceURI	string	The namespace URI of this Element/Attribute within XML instance documents/streams.
DeclaringType	Type	Type originally declaring the member (equal to ContainingType for non-inherited members)
ContainingType	Type	Type where this is a member of
DataType	Type	Data type of this member's content
Library	Library	Library containing this member's DataType
IsAttribute	boolean	True for attributes, false for elements
IsOptional	boolean	True if minOccurs = 0 or optional attribute
IsRequired	boolean	True if minOccurs > 0 or required attribute
IsFixed	boolean	True for fixed attributes, value is in Default property
IsDefault	boolean	True for attributes with default value, value is in Default property
IsNillable	boolean	True for nillable elements
IsUseQualified	boolean	True if NamespaceURI is not empty
MinOccurs	integer	minOccurs, as in schema. 1 for required attributes
MaxOccurs	integer	maxOccurs, as in schema. 0 for prohibited attributes, -1 for unbounded
Default	string	Default value

NativeBinding

This object represents the binding of a simple type to a native type in the target programming language, as specified by the "schemanativetype" map.

Property	Type	Description
ValueType	string	Native type
ValueHandler	string	Formatter class instance

Facets

This object represents all facets of a simple type. Inherited facets are merged with the explicitly declared facets. If a Length facet is in effect, MinLength and MaxLength are set to the same value.

Property	Type	Description
DeclaringType	Type	Type facets are declared on
Whitespace	string	"preserve", "collapse" or "replace"
MinLength	integer	Facet value
MaxLength	integer	Facet value
MinInclusive	integer	Facet value
MinExclusive	integer	Facet value
MaxInclusive	integer	Facet value

MaxExclusive	integer	Facet value
TotalDigits	integer	Facet value
FractionDigits	integer	Facet value
List	Facet collection	All facets as list

Facet

This object represents a single facet with its computed value effective for a specific type.

Property	Type	Description
LocalName	string	Facet name
NamespaceURI	string	Facet namespace
FacetType	string	one of "normalization", "lexicalspace", "valuespace-length", "valuespace-enum" or "valuespace-range"
DeclaringType	Type	Type this facet is declared on
FacetCheckerName	string	Name of facet checker (from schemafacet map)
FacetValue	string or integer	Actual value of this facet

Old object model (up to v2007)

The following objects are part of the old object model for code compatibility up to version 2007. They will be removed in a future release.

Namespace

Namespace abstraction. This object is part of the old object model for code compatibility up to version 2007.

Property	Type	Description
URI	string	The URI of this namespace
Prefix	string	The prefix of this namespace
ContainsPublicClasses	boolean	True if the Classes collection contains at least one non-internal Class object
Classes	Class collection	Collection of the classes in this namespace - step through them using foreach

Class

For every user-defined type declared in the schema (xsd) a class is generated. This object is part of the old object model for code compatibility up to version 2007.

Property	Type	Description
HasNamespace	boolean	True, if there is an associated Namespace object
Namespace	Namespace	The Namespace object this Class object is part of
NamespacePrefix	string	Prefix of the namespace in which the class is
NamespaceURI	string	URI of the namespace in which the class is
Name	string	Name of the type in the resulting file

SchemaName	string	Name of the type as in the original schema file
HasBaseObject	boolean	True if this type is derived from another type, which is also represented by a Class object.
BaseObject	Class	The base Class object if HasBaseObject is true
BaseNamespaceURI	string	Namespace URI of the base class
Base	string	Name of the base class
SchemaBase	string	Name of the base type as in the original schema file
BuiltInBase	string	Name of the root simple type
BuiltInSchemaBase	string	Name of the root simple type as in the original schema file
IsAbstract	boolean	True if this is an abstract type
IsSimpleType	boolean	True if this is a simple type
IsComplexFromSimpleType	boolean	True if this is a complex type and is derived from a simple type
IsComplexType	boolean	True if this is a complex type
IsSequence	boolean	True if the top-level group-type is "sequence"
IsChoice	boolean	True if the top-level group-type is "choice"
IsAll	boolean	True if the top-level group-type is "all"
Description	string	Description of this type. May contain line breaks.
IsGlobal	boolean	True if this type is declared globally in the schema
IsAnonymous	boolean	True if this type is declared locally in an element
IsInternal	boolean	True if this is the internal root pseudo-class that contains all global classes as members
Members	Member collection	A class representing a complexType contains one or more Members.
Facets	Facet collection	A class representing a simpleType or a complexType derived from a simpleType may contain Facets.

Member

Abstraction of a member-variable inside a class. Members are created for all children of a complex type. This object is part of the old object model for code compatibility up to version 2007.

Property	Type	Description
NamespaceURI	string	The namespace URI of this Element/Attribute within XML instance documents/streams.
Name	string	Name in the resulting file
SchemaName	string	Name as in the original schema file
XmlName	string	The name as it is expected to appear in XML instance documents/streams.
Type	string	The name of the class which represents the schema type
SchemaType	string	The schema type
TypeObject	Class	Generated class for this type. See explanation below *
HasTypeObject	boolean	True if TypeObject has a valid value
Description	string	Description of the Element/Attribute. Can contain line feeds.

IsBuiltinType	boolean	True if the type is a built-in schema type, e.g. string, unsignedInt, dateTime... It is the negation of HasTypeObject.
IsSimpleType	boolean	True if the type of this member is a simpleType
IsComplexFromSimpleType	boolean	True if the type of this member is a complex type and is derived from a simple type.
IsElement	boolean	True if this is an element
IsAttribute	boolean	True if this is an attribute
NodeType	string	"Element" or "Attribute"
MinOcc	integer	minOccurs, as in schema. 1 for required attributes
MaxOcc	integer	maxOccurs, as in schema. 0 for prohibited attributes
IsQualified	boolean	True if the form of the Element/Attribute is set to "qualified".
IsMixed	boolean	True if this element can have mixed content
HasTextValue	boolean	True if this member can contain text
Default	string	The default value defined in the schema
Fixed	string	The fixed value defined in the schema

* TypeObject:

For every type declared in the schema (xsd) a class is generated. All elements inside a complexType are generated as members of such classes. The types of these members can either be built-in types or user-defined types:

Built-in XML Schema types have no TypeObject, because they are mapped to predefined classes using the [map type](#) instruction.

For each member with a user-defined type, the TypeObject represents the class generated for this type.

Example:

```
<xs:complexType name="TypeA">
  <xs:sequence>
    <xs:element name="B" type="TypeB"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="TypeB"/>
```

CodeGen would create two classes:

```
class TypeA
{
  ...
  TypeB B;
}

class TypeB
{
  ...
}
```

Explanation:

When class "TypeA" is generated the member B is also generated. To find out the type of B, use the "TypeObject" method of the element which returns the class "TypeB". So we can get the name and use it there.

Facet

This class consists of the constraint itself, and several boolean variables. This object is part of the old object model for code compatibility up to version 2007.

The boolean variables tell you the kind of constraint you're currently dealing with. The names of the Boolean variables are identical to those used in the schema specification.

Property	Type	Description
Constraint	string	Holds the value of the facet
IsLength	boolean	
IsMinLength	boolean	
IsMaxLength	boolean	
IsMinInclusive	boolean	
IsMinExclusive	boolean	
IsMaxInclusive	boolean	
IsMaxExclusive	boolean	
IsTotalDigits	boolean	
IsFractionDigits	boolean	
IsWhiteSpace	boolean	
IsPattern	boolean	
IsEnumeration	boolean	
Enumeration	Enumeration collection	Holds a collection of Enumeration objects if this facet is of type enumeration.
Pattern	Pattern collection	Holds a collection of Enumeration objects if this facet is of type pattern.

Enumeration

Abstraction of an enumeration entry inside a facet. This object is part of the old object model for code compatibility up to version 2007.

Property	Type	Description
Index	integer	Holds the index of this enumeration value, starting with 0
Value	string	Holds an enumeration value

Pattern

Abstraction of a pattern entry inside a facet. This object is part of the old object model for code compatibility up to version 2007.

Property	Type	Description
Index	integer	Holds the index of this pattern value, starting with 0
Value	string	Holds a pattern value

20.8 Error Codes

Operating System Error Codes

- 201 File not found: '%s'
- 202 Cannot create file '%s'
- 203 Cannot open file '%s'
- 204 Cannot copy file '%s' to '%s'

Syntax Error Codes

- 401 Keyword expected
- 402 '%s' expected
- 403 No output file specified
- 404 Unexpected end of file
- 405 Keyword not allowed

Runtime Error Codes

- 501 Unknown variable '%s'
- 502 Redefinition of variable '%s'
- 503 Variable '%s' is not a container
- 504 Unknown property '%s'
- 505 Cannot convert from %s to %s
- 507 Unknown function
- 508 Function already defined
- 509 Invalid parameter
- 510 Division by zero
- 511 Unknown method
- 512 Incorrect number of parameters
- 513 Stack overflow

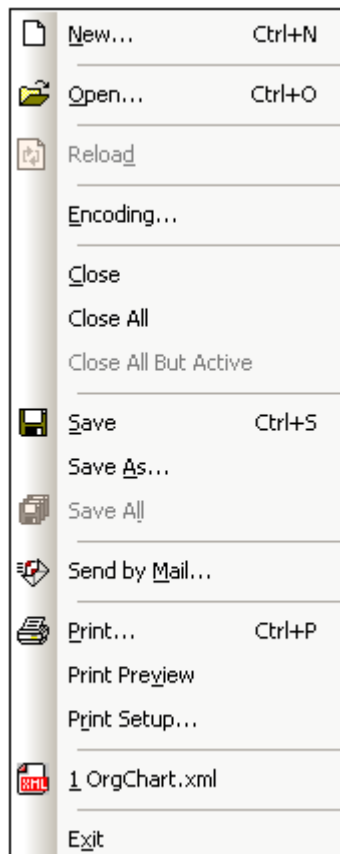
21 User Reference

The **User Reference** section contains a complete description of all XMLSpy menu commands and explains their use in general. We have tried to be comprehensive. If, however, you have questions which are not covered in the User Reference or other parts of this documentation, please look up the FAQs and Discussion Forums on the Altova website. If you cannot find a suitable answer at these locations, please do not hesitate to contact the [Altova Support Center](#).

Standard Windows commands, such as (**Open**, **Save**, **Cut**, **Copy** and **Paste**) are in the [File](#) and [Edit](#) menus. These menus additionally contain XML- and Internet-related commands.

21.1 File Menu

The **File** menu contains commands relevant to manipulating files, in the order common to most Windows software products.

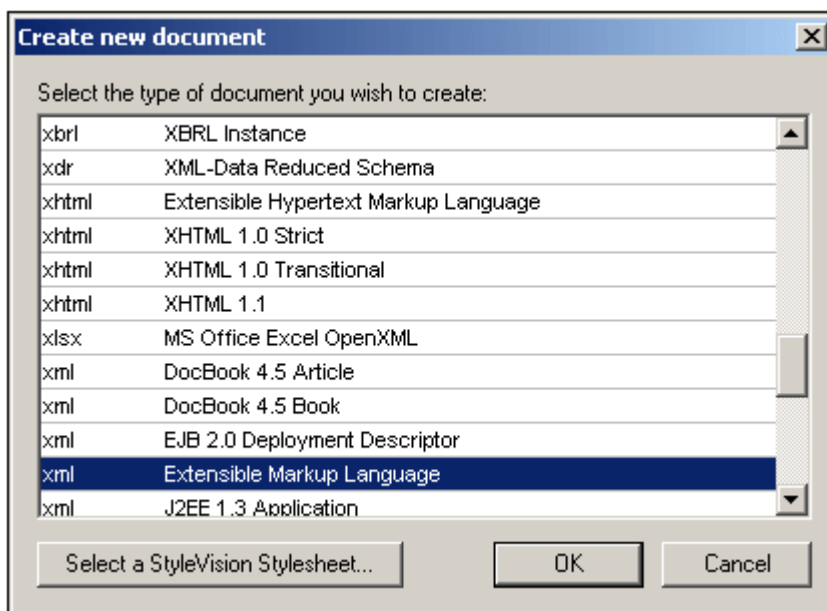


In addition to the standard [New](#), [Open](#), [Save](#), [Print](#), [Print Setup](#), and [Exit](#) commands, XMLSpy offers a range of XML- and application-specific commands.

21.1.1 New



The **New** command is used to create a new document. Clicking **New** opens the Create New Document dialog, in which you can select the type of document you wish to create. If the document type you wish to create is not listed, select XML and change the file extension when you save the file. Note that you can add new file types to the list in this dialog using the [Tools | Options | File types tab](#).

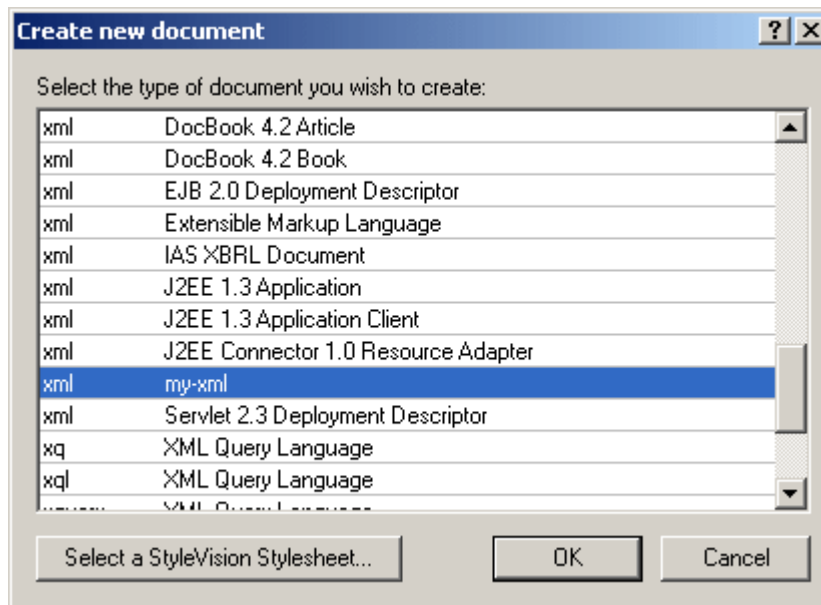


Creating templates for new documents

You can create multiple templates for various file types. These templates can then be opened directly from the Create New Document dialog and edited. To create your own template so that it appears in the list of documents in the Create New Document dialog, you first create the template document and then save it to the folder that contains all the templates.

Do the following:

1. Open the **XMLSpy\Template** folder using Windows Explorer or your preferred navigation tool, and select a rudimentary template file from among the files named **new. xxx** (where . xxx is a file extension, such as . xml and . xslt).
2. Open the file in XMLSpy, and modify the file as required. This file will be the template file.
3. When you are done, select **File | Save as...** to save the file back to the `\Template` folder with a suitable name, say `my-xml. xml`. You now have a template called `my-xml`, which will appear in the list of files in the Create New Document dialog.



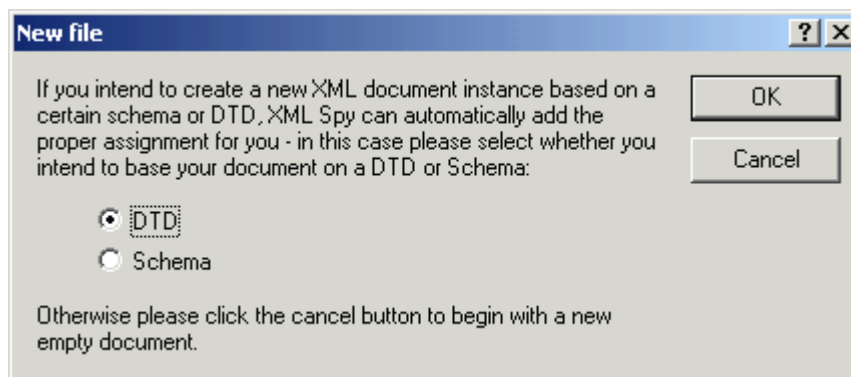
4. To open the template, select **File | New**, and then the template (`my-xml`, in this case).

Please note: To delete a template, delete the template file from the template folder.

Assigning a DTD/XML Schema to a new XML document

When you create a new document of a certain type that is based on a standard schema (DTD or XML Schema), the document is automatically opened with the correct DTD or XML Schema association. For example, an XHTML file will be opened with the DTD `http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd` associated with it. And an XML Schema (.xsd) file is associated with the `http://www.w3.org/2001/XMLSchema` schema document.

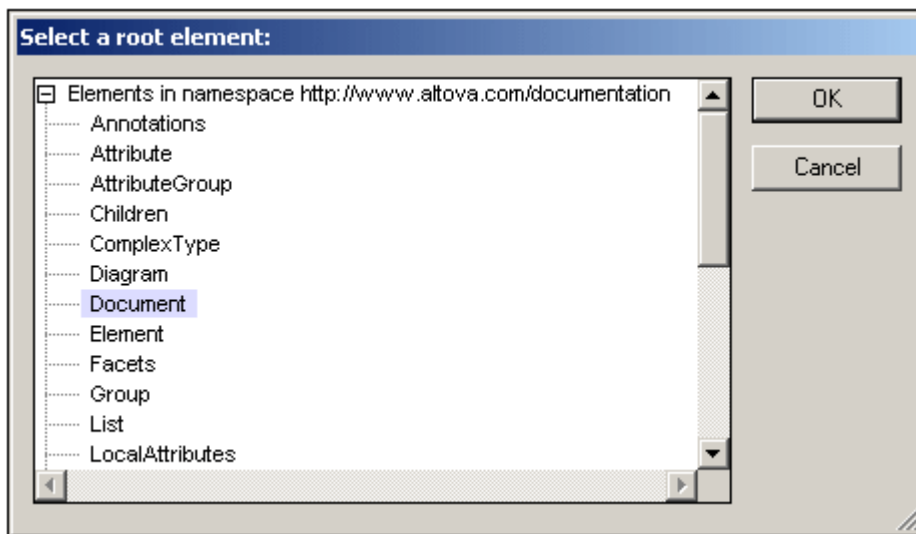
If you are creating a new file for which the schema is not known (for example, an XML file), then you are prompted to associate a schema (DTD or XML Schema) with the document that is to be created.



If you choose to associate a DTD or XML Schema with your document, clicking **OK** in the New File dialog enables you to browse for the schema. Clicking **Cancel** in this dialog will create a new file that is not associated with any schema.

Specifying the document element of a new XML document

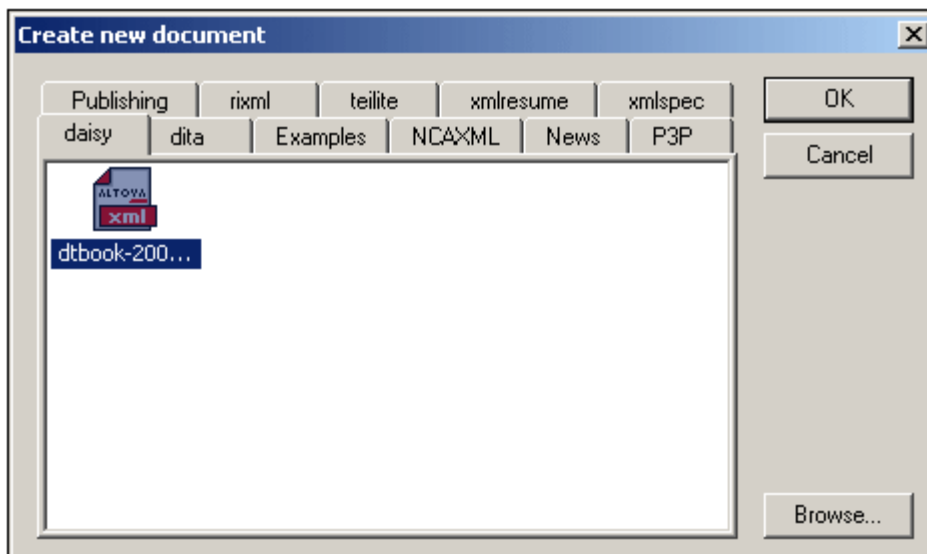
If you select an XML Schema, there can be more than one global element in it, all of which are potential document (or root) elements. You can select which of these is to be the root element of the XML document in the Select a Root Element dialog, which pops up if you select Schema in the New File dialog and if the XML Schema has more than one global element.



The new XML document is created with this element as its document element.

Assigning a StyleVision Power Stylesheet when creating a new document

When a new XML document is created, you can associate a StyleVision Power Stylesheet (. sps file) to view the document in Authentic View. In the Create New Document dialog (see *screenshot above*), when you click the **Select StyleVision Stylesheet**, the Create New Document dialog (*shown below*) appears.



You can browse for the required StyleVision Power Stylesheet in the folder tabs displayed in the New dialog. Alternatively, you can click the **Browse** button to navigate for and select the StyleVision Power Stylesheet. The tabs that appear in the New dialog correspond to folders in the `sps/Template` folder of your application folder.

Creating new XBRL taxonomies with the XBRL Taxonomy Wizard

In the Create a New Document dialog, if XBRL Taxonomy Schema (.xsd) is selected, then a wizard guides you through the steps for creating a new XBRL taxonomy. This wizard is described in the [XBRL section](#) of the documentation.

21.1.2 Open

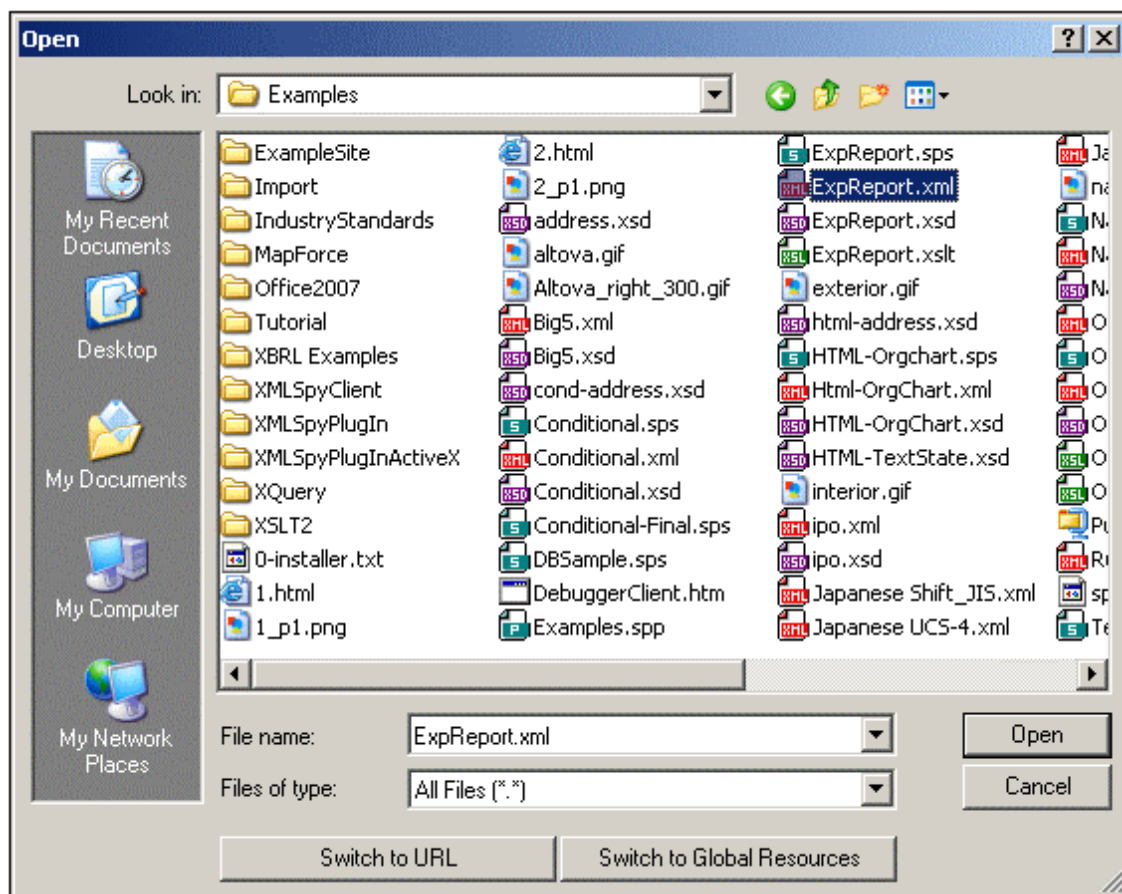


Ctrl+O

The **Open** command pops up the familiar Windows Open dialog, and allows you to open any XML-related document or text document. In the Open dialog, you can select more than one file to open. Use the Files of Type combo box to restrict the kind of files displayed in the dialog box. (The list of available file types can be configured in the File Types tab of the Options dialog ([Tools | Options](#)).) When an XML file is opened, it is checked for well-formedness. If the file is not well-formed, you will get a file-not-well-formed error. Fix the error and select the menu command **XML | Check Well-Formedness (F7)** to recheck. If you have opted for automatic [validation upon opening](#) and the file is invalid, you will get an error message. Fix the error and select the menu command **XML | Validate XML (F8)** to revalidate.

Selecting files via URLs and Global Resources

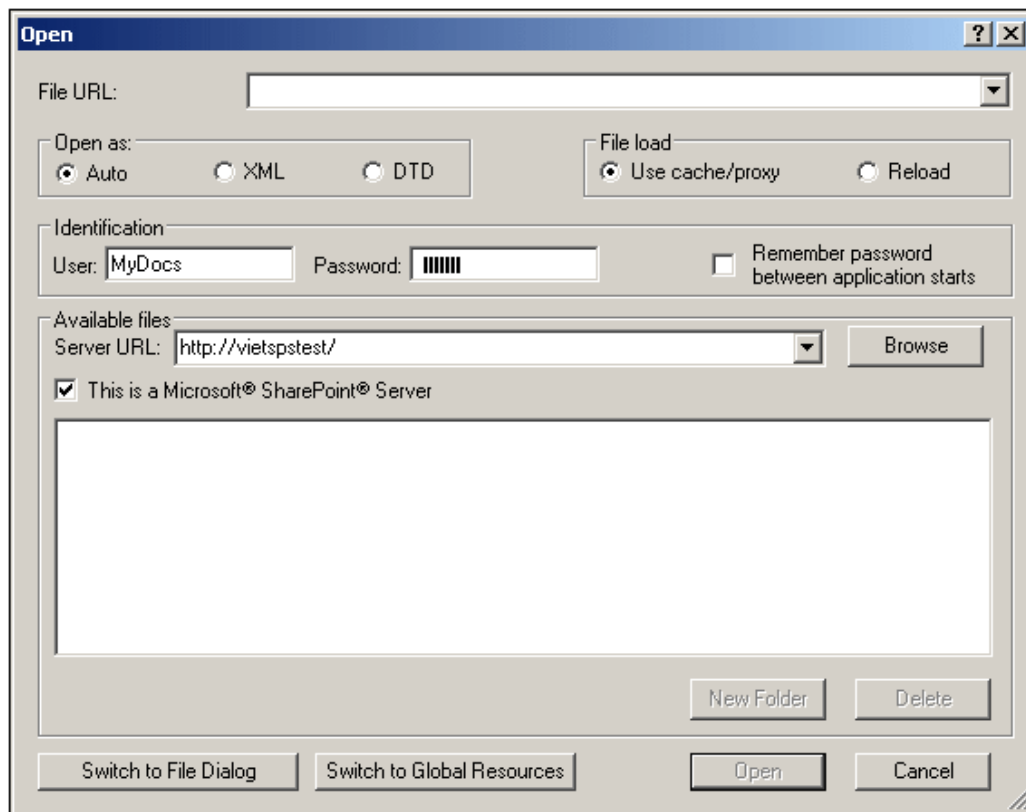
In several File Open and File Save dialogs, you can choose to select the required file or save a file via a URL or a global resource (*see screenshot below*). Select the **Switch to URL** or **Switch to Global Resource** to go to one of these selection processes.



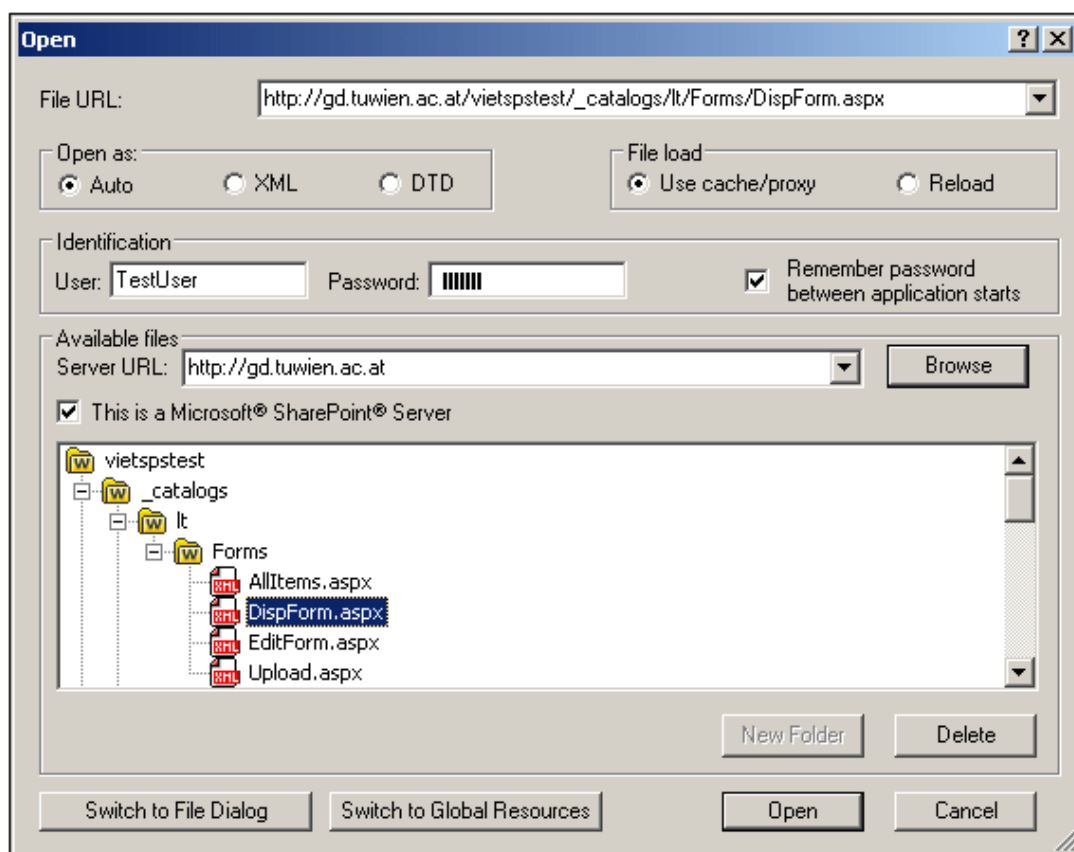
Selecting files via URLs

To select a file via a URL, do the following:

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



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



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

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

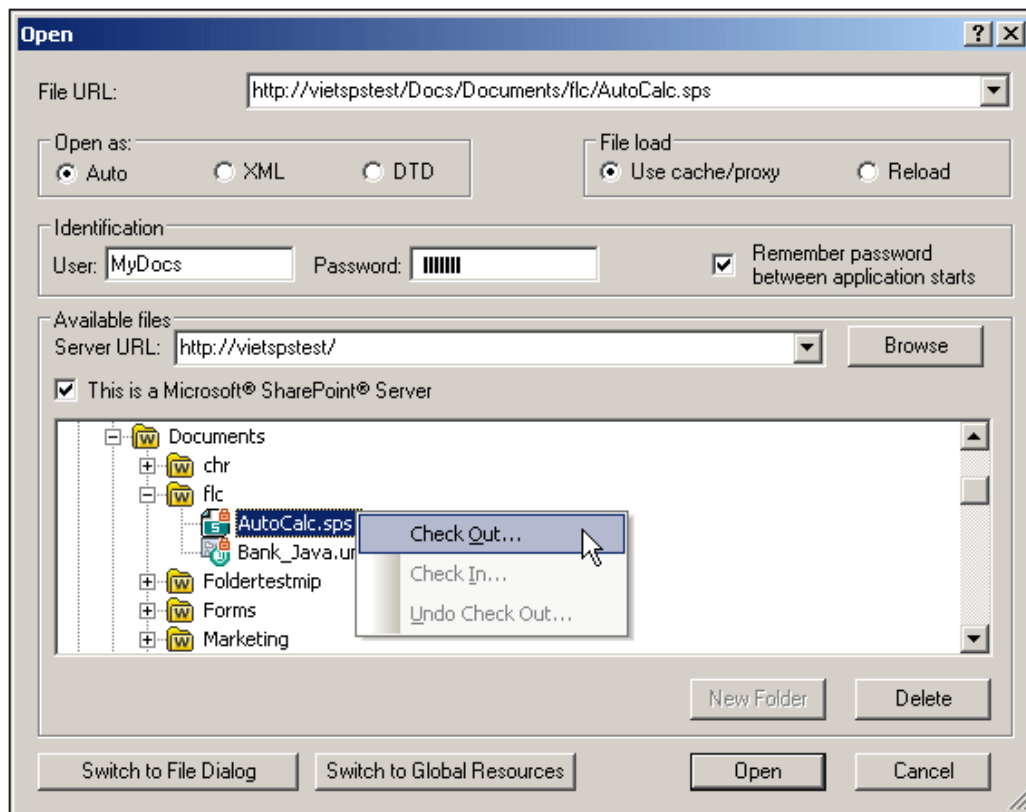
Note: The Browse function is only available on servers which support WebDAV and on Microsoft SharePoint Servers. The supported protocols are FTP, HTTP, and HTTPS.

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

Microsoft® SharePoint® Server Notes

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

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

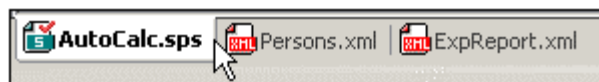


Right-clicking a file pops up a context menu containing commands available for that file (*screenshot above*).

- The various file icons are shown below:

	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.

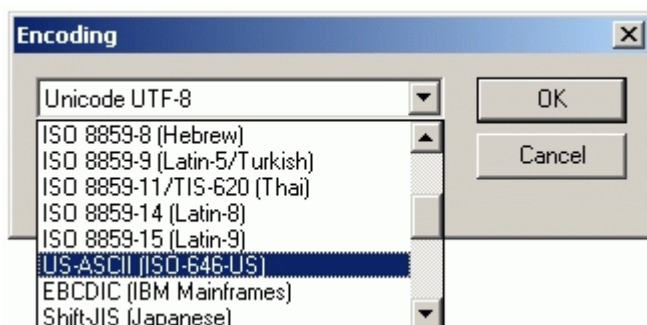
21.1.3 Reload



The **Reload** command allows you to reload open documents. This is useful if an open document has been modified outside XMLSpy. If a modification occurs, XMLSpy asks whether you wish to reload the file. If you reload, then any changes you may have made to the file since the last save will be lost. This option can be changed in the Options dialog ([Tools | Options](#)).

21.1.4 Encoding

The **Encoding** command lets you view the current encoding of the active document (XML or non-XML) and to select a different encoding with which the active document will be saved the next time.



In XML documents, if you select a different encoding than the one in use before, the encoding specification in the XML declaration will be adjusted accordingly. For two-byte and four-byte character encodings (UTF-16, UCS-2, and UCS-4) you can also specify the byte-order to be used for the file. Another way to change the encoding of an XML document is to directly edit the encoding attribute of the document's XML declaration.

Default encodings for existing and new XML and non-XML documents can be set in the [Encoding tab of the Options dialog](#).

Note: When saving a document, XMLSpy automatically checks the encoding specification and opens a dialog box if it cannot recognize the encoding entered by the user. Also, if your document contains characters that cannot be represented in the selected encoding, you will get a warning message when you save your file.

21.1.5 Close, Close All, Close All But Active

The **Close** command closes the active document window. If the file was modified (indicated by an asterisk * after the file name in the title bar), you will be asked if you wish to save the file first.

The **Close All** command closes all open document windows. If any document has been modified (indicated by an asterisk ***** after the file name in the title bar), you will be asked if you wish to save the file first.

The **Close All But Active** command closes all open document windows except the active document window. If any document has been modified (indicated by an asterisk ***** after the file name in the title bar), you will be asked if you wish to save the file first.

21.1.6 Save, Save As, Save All

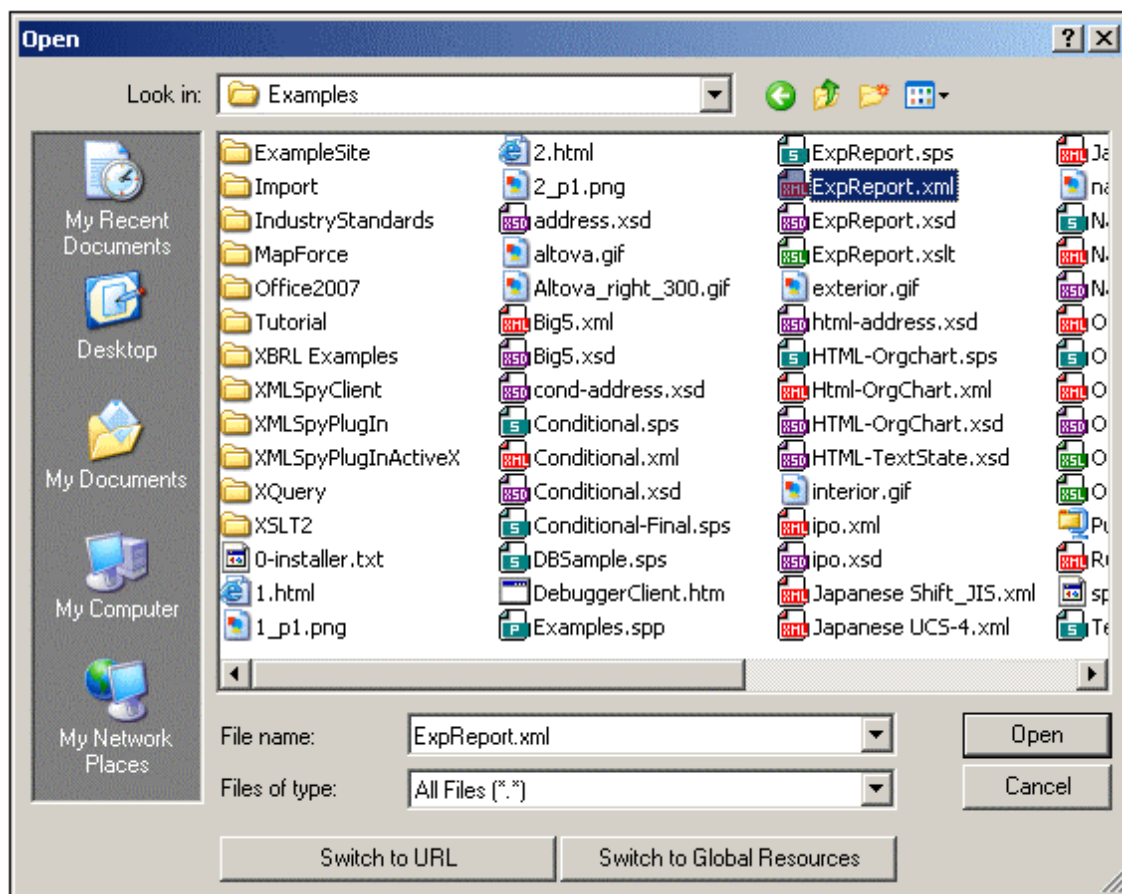
The **Save (Ctrl+S)**  command saves the contents of the active document to the file from which it has been opened. When saving a document, the file is automatically [checked for well-formedness](#). The file will also be validated automatically if this option has been set in the File tab of the Options dialog ([Tools | Options](#)). The XML declaration is also checked for the [encoding](#) specification, and this encoding is applied to the document when the file is saved.

The **Save As** command pops up the familiar Windows Save As dialog box, in which you enter the name and location of the file you wish to save the active file as. The same checks and validations occur as for the **Save** command.

The **Save All**  command saves all modifications that have been made to any open documents. The command is useful if you edit multiple documents simultaneously. If a document has not been saved before (for example, after being newly created), the Save As dialog box is presented for that document.

Selecting files via URLs and Global Resources

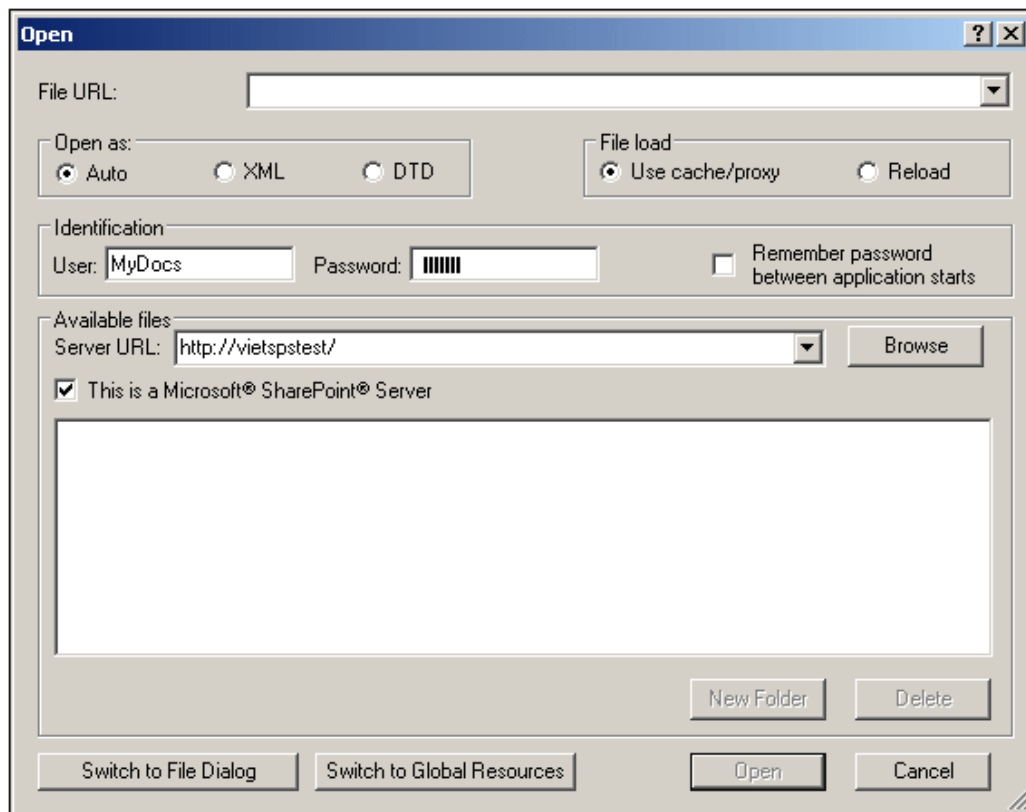
In several File Open and File Save dialogs, you can choose to select the required file or save a file via a URL or a global resource (*see screenshot below*). Select the **Switch to URL** or **Switch to Global Resource** to go to one of these selection processes.



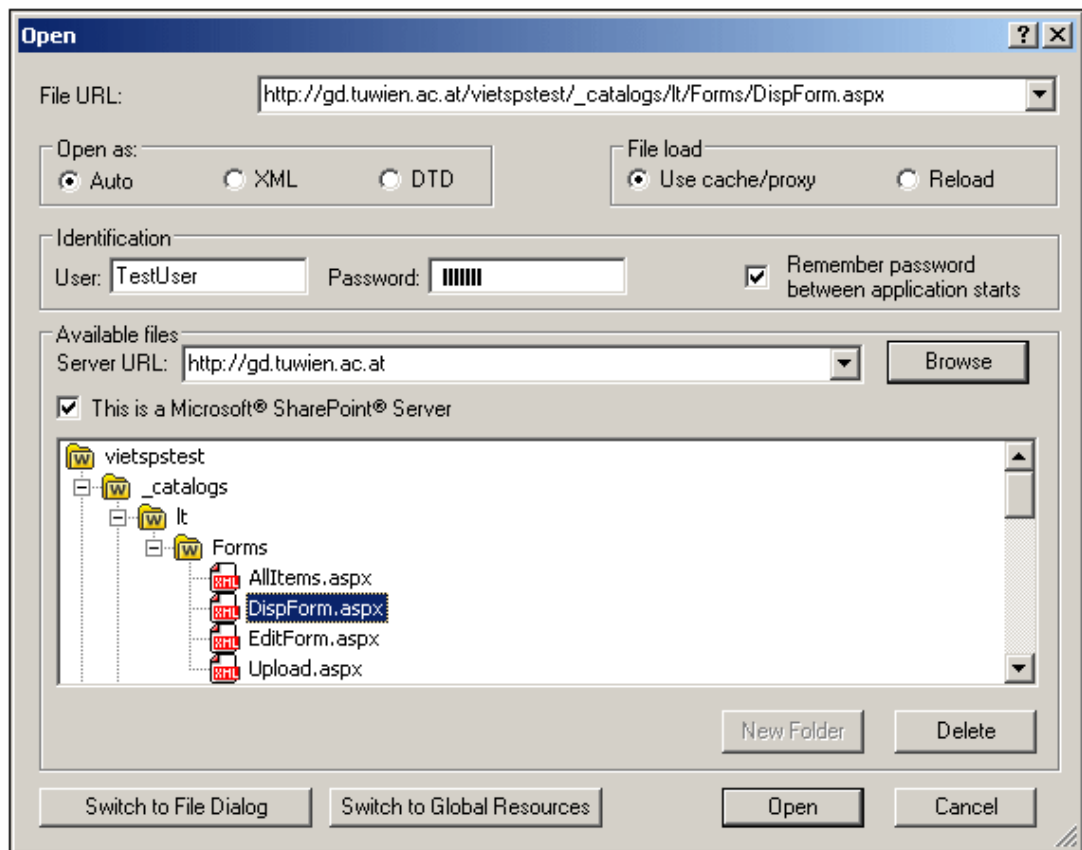
Selecting files via URLs

To select a file via a URL, do the following:

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



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



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

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

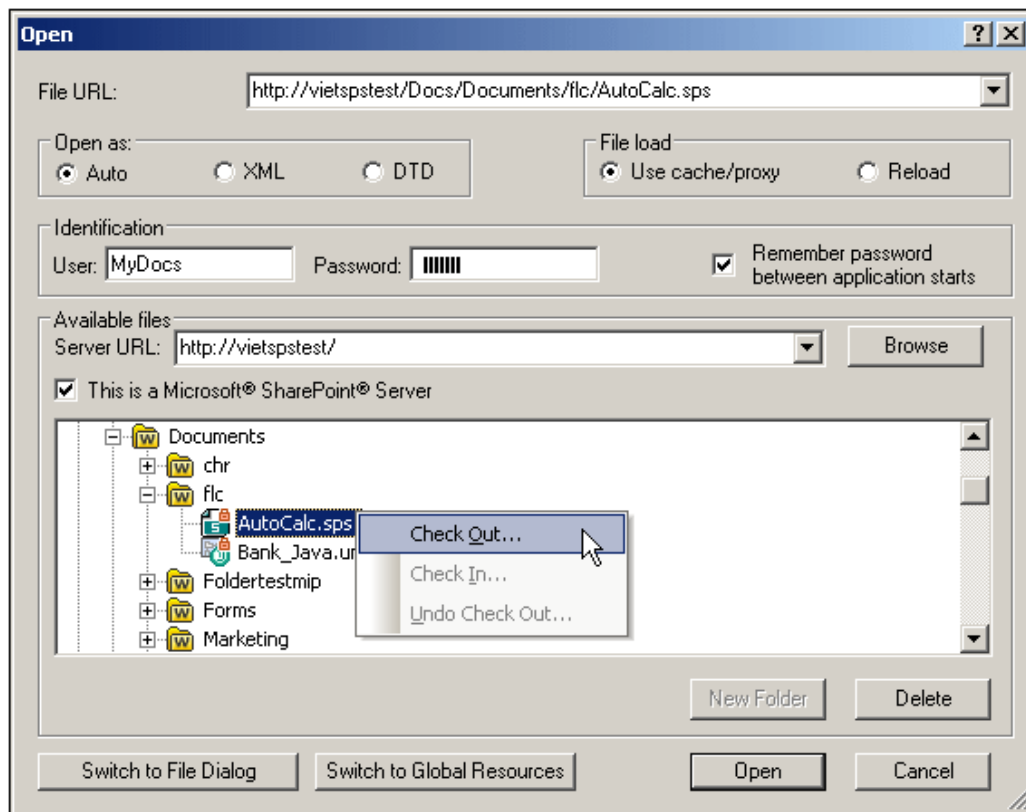
Note: The Browse function is only available on servers which support WebDAV and on Microsoft SharePoint Servers. The supported protocols are FTP, HTTP, and HTTPS.

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

Microsoft® SharePoint® Server Notes

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

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

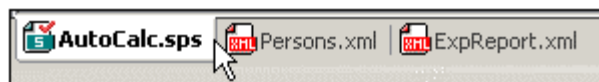


Right-clicking a file pops up a context menu containing commands available for that file (*screenshot above*).

- The various file icons are shown below:

	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.

21.1.7 Send by Mail



The **Send by Mail...** command lets you send XML document/s or selections from an XML document by e-mail. You can do any of the following:

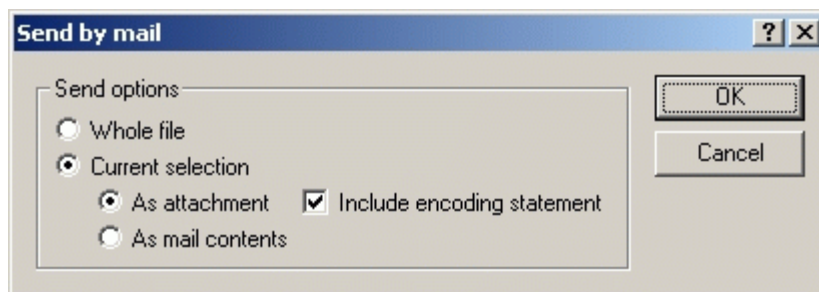
- Send an XML document as an attachment or as the content of an e-mail.
- Send a selection in an XML document as an attachment or as the content of an e-mail.
- Send a group of files (selected in the Project Window) as an attachment to an e-mail.
- Send a URL (selected in the Project Window) as an attachment or as a link.

Please note: To use this function you must have a MAPI compliant e-mail system.

Sending documents and document fragments

To send an XML document:

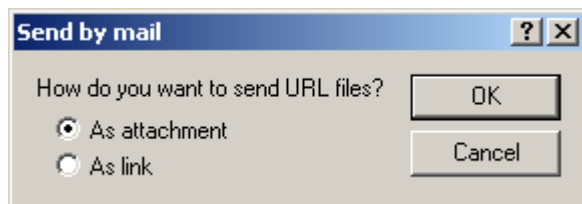
1. Make that document the active document in the Main Window. If you wish to send a selection or a group of files from a project, make the selection in the document or select the required files in the Project.
2. Click **Send by Mail...**. The following dialog opens:



3. Make the required choices and click **OK**. The selected documents/contents/URLs are attached to the e-mail or content is inserted into the e-mail.

Sending URLs by mail

To send one or more URLs, select the URLs in the Project Window, and click **Send by Mail...**. The following dialog opens:



Select how the URL is to be sent and click **OK**.

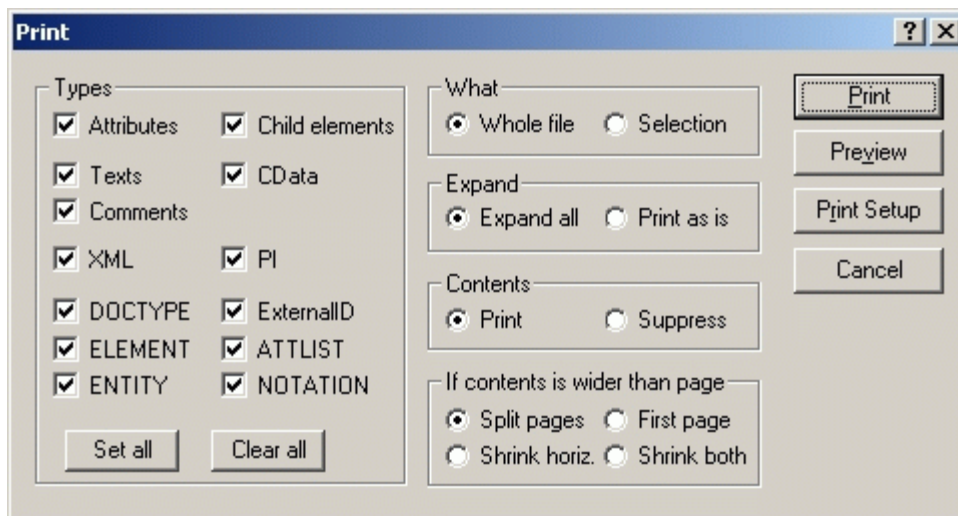
21.1.8 Print



Ctrl+P

The **Print** command opens the Print dialog box, in which you can select printer options. The currently active document, as seen in the current view, can then be printed.

Clicking the **Print** command in Grid View opens a Print options dialog (*screenshot below*), which enables you to set printing options for the XML document. Clicking **Print** in this dialog takes you to the Print dialog for printer options.



The available options for Grid View printing are described below:

- In the Types pane, you select the items you wish to have appear in the output.
- For the What option, you specify whether the current selection or the entire file is to be printed.
- The Expand option allows you to print the document as is, or with all descendant elements expanded fully.
- The Contents option enables you to choose between printing contents of all nodes or printing node names only.
- In the If Contents Are Wider Than Page pane, you select what to do if contents are wider than the page. The Split Pages option prints the entire document at normal size, splitting contents over pages both horizontally and vertically. The pages could then be glued together to form a poster. The First Page option prints only the first, left-hand page of the print area. The area that overflows horizontally is not printed. This option is useful if most of the important information in your Grid View of the document is contained on the left side. The Shrink Horizontally option reduces the size of the output (proportionally) until it fits horizontally on the page; the document may run on for several pages. The Shrink Both option shrinks the document in both directions until it fits exactly on one sheet.
- The Print button prints the document with the selected options.
- The Preview button opens a print preview window that lets you view the final output before committing it to paper.
- The Print Setup button opens the Print Setup dialog box and allows you to adjust the paper format, orientation, and other printer options for this print job only. Also see the [Print Setup](#) command.

Note: You can change column widths in Grid View to optimize the print output.

Program logo

If you have a purchased license, you can turn off the program logo, copyright notice, and registration details when printing a document from XMLSpy. This option is available in the [View tab of the Options dialog](#).

WSDL and XBRL designs

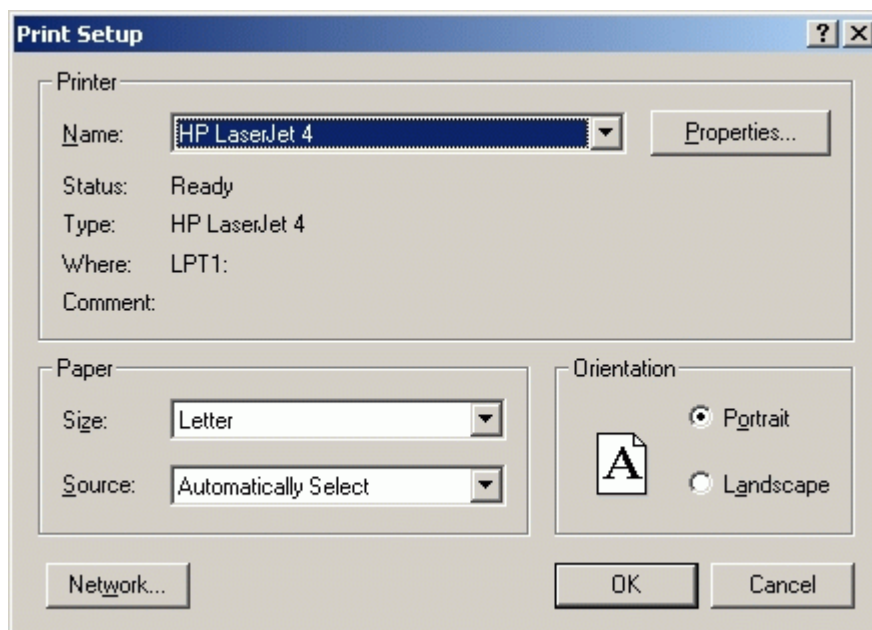
Graphical views of WSDL and XBRL documents, as seen on screen, can also be printed using the **Print** command.

21.1.9 Print Preview, Print Setup

The **Print Preview** command opens the Print dialog box. Click the **Preview** button to display a print preview of the currently active document. In Print Preview mode, the Print Preview toolbar at top left of the preview window provides print- and preview-related options.

The preview can be magnified or miniaturized using the **Zoom In** and **Zoom Out** buttons. When the page magnification is such that an entire page length fits in a preview window, then the **One Page / Two Page** button toggles the preview to one or two pages at a time. The **Next Page** and **Previous Page** buttons can be used to navigate among the pages. The toolbar also contains buttons to print all pages and to close the preview window.

The **Print Setup** command, displays the printer-specific Print Setup dialog box, in which you specify such printer settings as paper format and page orientation. These settings are applied to all subsequent print jobs.



The screenshot above shows the Print Setup dialog in which an HP LaserJet 4 printer attached to a parallel port (LPT1) is selected.

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**.

21.1.10 Recent Files, Exit

The **File** menu displays a list of the nine most recently used files, with the most recently opened file shown at the top of the list. You can open any of these files by clicking its name. To open a file in the list using the keyboard, press **ALT+F** to open the **File** menu, and then press the number of the file you want to open.

The **Exit** command is used to quit XMLSpy. If you have any open files with unsaved changes, you are prompted to save these changes. XMLSpy also saves modifications to program settings and information about the most recently used files.

21.2 Edit Menu

The **Edit** menu contains commands for editing documents in XMLSpy.

	U <u>ndo</u>	Ctrl+Z
	R <u>edo</u>	Ctrl+Y
	C <u>ut</u>	Ctrl+X
	C <u>opy</u>	Ctrl+C
	P <u>aste</u>	Ctrl+V
	D <u>elete</u>	Delete
	Copy as XML Text	
	Copy as Structured Text	
	C <u>opy</u> X <u>P</u> ath	
	C <u>opy</u> X <u>P</u> ointer	
	I <u>nsert</u>	▶
	P <u>retty-Print</u> XML Text	
	S <u>elect</u> <u>A</u> ll	Ctrl+A
	F <u>ind</u> ...	Ctrl+F
	F <u>ind</u> <u>N</u> ext	F3
	R <u>eplace</u> ...	Ctrl+H
	F <u>ind in Files</u> ...	
	I <u>nsert/Remove</u> Bookmark	Ctrl+F2
	R <u>emove All</u> Bookmarks	Ctrl+Shift+F2
	G <u>o to Next</u> Bookmark	F2
	G <u>o to Previous</u> Bookmark	Shift+F2
	C <u>omment In/Out</u>	Ctrl+K

In addition to the standard [Undo](#), [Redo](#), [Cut](#), [Copy](#), [Paste](#), [Delete](#), [Select All](#), [Find](#), [Find next](#) and [Replace](#) commands, XMLSpy offers special commands to:

- [copy the selection to the clipboard as XML-Text](#),
- [copy as structured text](#)
- [copy an XPath selector to the selected item](#) to the clipboard.
- insert and remove bookmarks, and to navigate to bookmarks.

21.2.1 Undo, Redo

The **Undo** (Ctrl+Z) command  contains support for unlimited levels of Undo. Every action can be undone and it is possible to undo one command after another. The Undo history is retained

after using the Save command, enabling you go back to the state the document was in before you saved your changes.

The **Redo (Ctrl+Y)** command  allows you to redo previously undone commands, thereby giving you a complete history of work completed. You can step back and forward through this history using the Undo and Redo commands.

21.2.2 Cut, Copy, Paste, Delete

The **Cut (Shift+Del or Ctrl+X)** command  copies the selected text or items to the clipboard and deletes them from their present location.

The **Copy (Ctrl+C)** command  copies the selected text or items to the clipboard. This can be used to duplicate data within XMLSpy or to move data to another application.

Please note: There are two different commands for copying elements to the clipboard in textual form: [Copy as XML-Text](#) and [Copy as Structured Text](#). You can use the [Editing tab on the Tools | Options dialog](#) to choose which of these two operations should be performed when using the copy command.

The **Paste (Ctrl+V)** command  inserts the contents of the clipboard at the current cursor position.

The **Delete (Del)** command  deletes the currently selected text or items without placing them in the clipboard.

21.2.3 Copy as XML Text

The **Copy as XML-Text** command lets you exchange data easily with other products that allow data manipulation on the XML source layer. While editing your document in Grid View, you may occasionally want to copy some elements to the clipboard in their XML-Text representation:

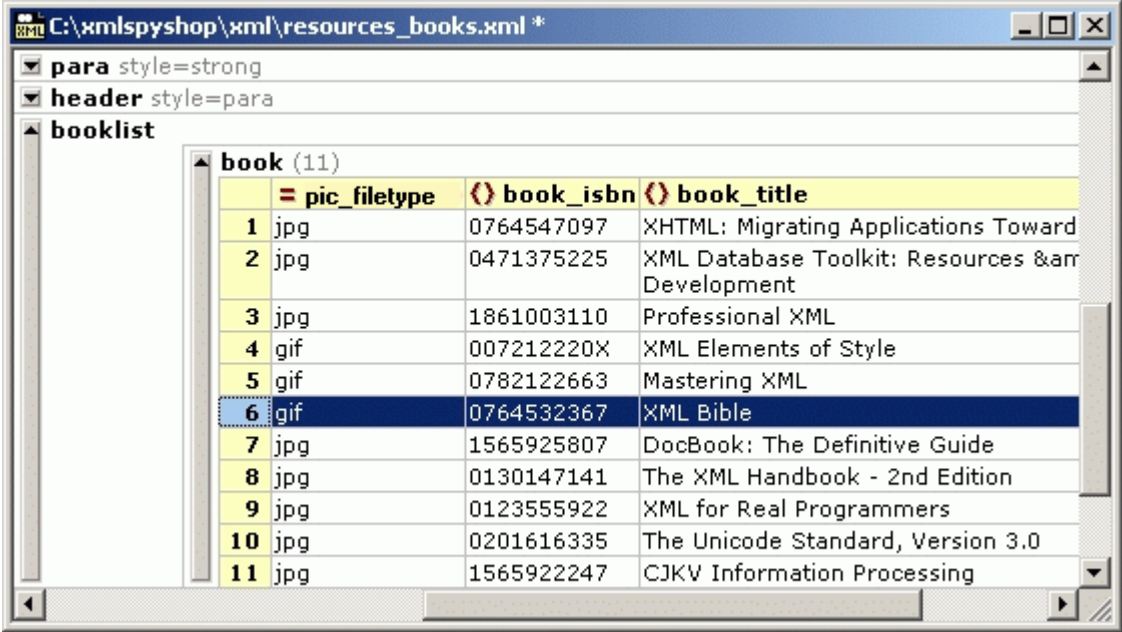
```
<row>
  <para align="left">
    <bold>Check the FAQ</bold>
  </para>
  <para>
    <link mode="internal">
      <link_section>support</link_section>
      <link_subsection>faq30</link_subsection>
      <link_text>XMLSPY 4.0 FAQ</link_text>
    </link>
    <link mode="internal">
      <link_section>support</link_section>
      <link_subsection>faq25</link_subsection>
      <link_text>XMLSPY 3.5 FAQ</link_text>
    </link>
  </para>
</row>
```

The **Copy as XML-Text** command automatically formats text using the currently active settings

for saving a file. These settings can be modified in the Save File section of the File tab of the Options dialog (**Tools | Options**).

21.2.4 Copy as Structured Text

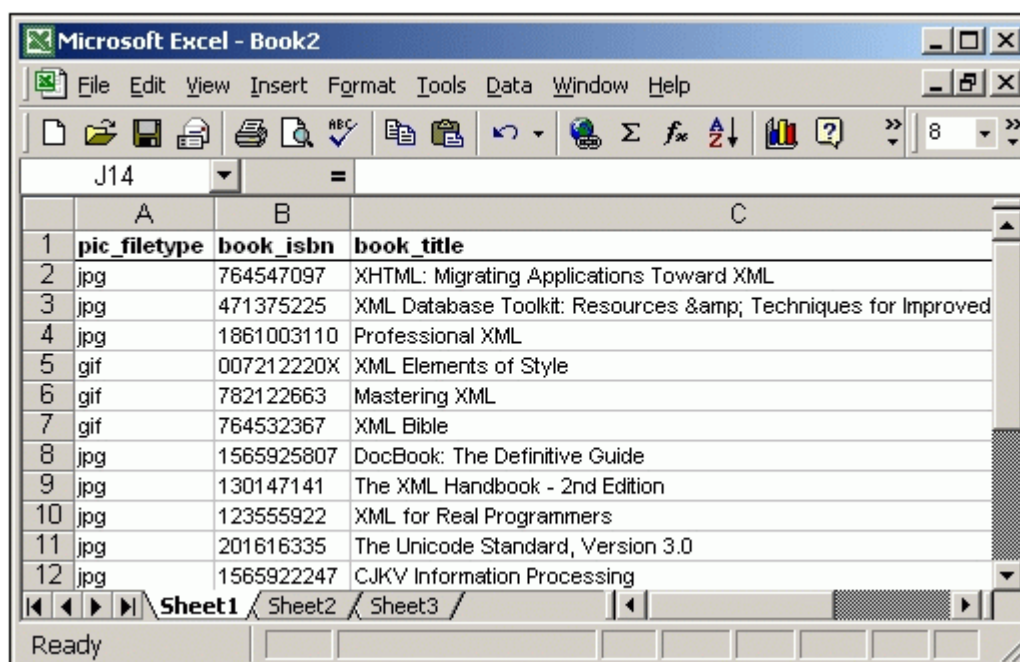
The **Copy as Structured Text** command copies elements to the clipboard as they appear on screen. This command is useful for copying table-like data from Grid View. The copied data can be used within XMLSpy as well as in third-party products, enabling you to transfer XML data to spreadsheet-like applications (such as Microsoft Excel).



The screenshot shows the XMLSpy application window with the file 'C:\xmlspyshop\xml\resources_books.xml'. The 'booklist' element is expanded, showing a table with 11 rows. The table has three columns: 'pic_filetype', 'book_isbn', and 'book_title'. Row 6 is selected.

	pic_filetype	book_isbn	book_title
1	jpg	0764547097	XHTML: Migrating Applications Toward
2	jpg	0471375225	XML Database Toolkit: Resources & Development
3	jpg	1861003110	Professional XML
4	gif	007212220X	XML Elements of Style
5	gif	0782122663	Mastering XML
6	gif	0764532367	XML Bible
7	jpg	1565925807	DocBook: The Definitive Guide
8	jpg	0130147141	The XML Handbook - 2nd Edition
9	jpg	0123555922	XML for Real Programmers
10	jpg	0201616335	The Unicode Standard, Version 3.0
11	jpg	1565922247	CJKV Information Processing

If you copy this table and paste it into Excel, the data will appear in the following way:



Please note: The results of this command depend on the way the information is currently laid out on screen. For example, if the XML fragment used as an example for the [Copy as XML-Text](#) command were in the Table View of Grid View, the copied text would result in the following:

```
row
  para
    align    bold    link
    left     Check the FAQ
      link
        mode      link_section  link_subsection  link_text
        internal  support       faq30           XMLSPY 3.5 FAQ
        internal  support       faq25           XMLSPY 2.5 FAQ
```

In normal Grid View, the data copied to the clipboard would look as below.

```
row
  para
    align  left
    bold   Check the FAQ
  para
    link
      mode      internal
      link_section  support
      link_subsection  faq30
      link_text       XMLSPY 3.5 FAQ
    link
      mode      internal
      link_section  support
      link_subsection  faq25
      link_text       XMLSPY 2.5 FAQ
```

21.2.5 Copy XPath

The **Copy XPath** command is available in Text View and Grid View, and creates an XPath expression that selects the currently selected node/s and copies the expression to the clipboard. This enables you to paste the expression into a document (for example, in an XSLT document). All expressions start from the document root.

The XPath expression is resolved differently in Grid View and Text View. In Grid View, if a single element is highlighted, the XPath expression will select not that specific element but all elements of that name at that hierarchical level of the document. In Text View that specific element is selected. For example, if an element called `LastName` of the third `Person` element of the second `Company` element is selected, the XPath expressions would be as follows:

- Grid View: `/Companies/Company/Person/LastName`
- Text View: `/Companies/Company[ 2] /Person[ 3] /LastName`

Note: In Grid View the **Copy XPath** command can also be accessed via the context menu.

21.2.6 Copy XPointer

The **Copy XPointer** command is available in Text View and Grid View. It creates an element() scheme XPointer for the currently selected node/s and copies it to the clipboard. This enables you to paste the XPointer into a document (for example, in the `xpointer` attribute of an `XInclude` element in an XML document).

The element() scheme of XPointer returns results in the form `element( /1/3 )`, which selects the third child of the document element (or root element). You should note the following points:

- Attributes cannot be represented using the element() scheme. If an attribute is selected, the following happens: In Grid View, the **Copy XPointer** command is disabled; in Text View, the XPointer of the element "parent" of that attribute is generated.
- Multiple elements cannot be selected. If selected in Grid View, the **Copy XPointer** command is disabled. In Text View, the XPointer of the parent element of the selection is generated.

Note: In Grid View the **Copy XPointer** command can also be accessed via the context menu.

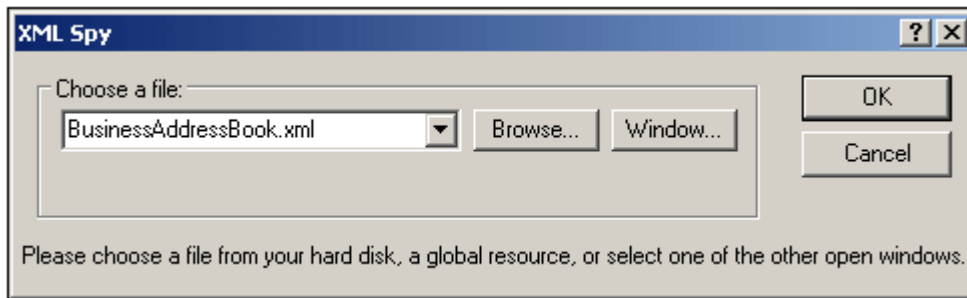
21.2.7 Insert

Mousing over or selecting the **Insert** command rolls out a submenu with three commands, which are described below:

- [Insert File Path](#)
- [Insert XInclude](#)
- [Insert Encoded External File](#)

Insert File Path

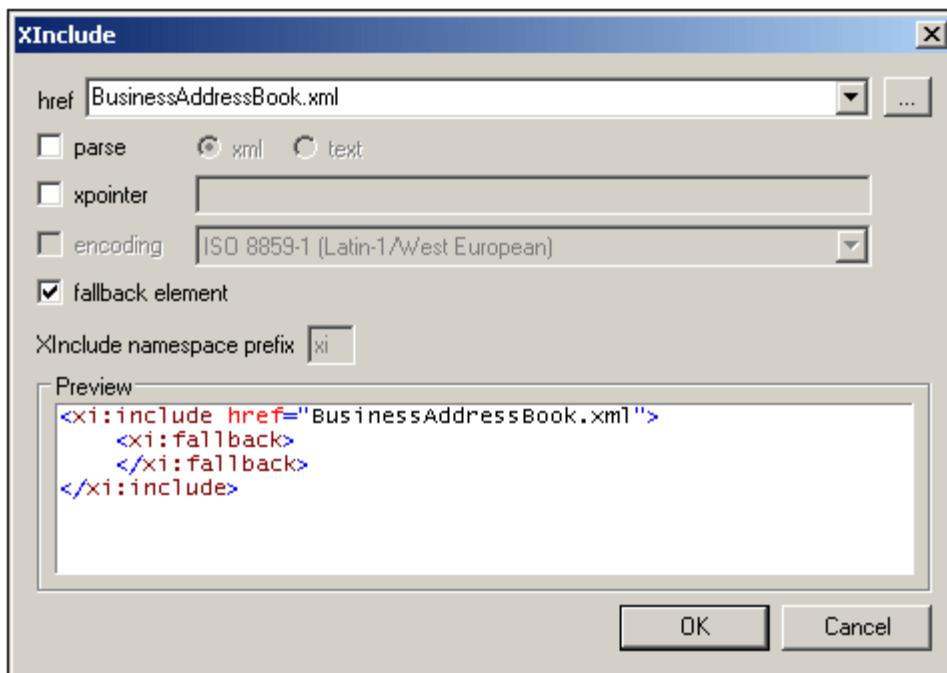
The **File Path** command is enabled in the Text View and Grid View of documents of any file type. Using it, you can insert the path to a file at the cursor selection point. Clicking the command pops up a dialog (*screenshot below*) in which you select the required file.



The required file can be selected in one of the following ways: (i) by browsing for the file, URL, or global resource (use the [Browse](#) button); (ii) by selecting the window in which the file is open (the **Window** button). When done, click **OK**. The path to the selected file will be inserted in the active document at the cursor selection point.

Insert XInclude

The **XInclude** command is available in Text View and Grid View, and enables you to insert a new XInclude element at the cursor selection point in Text View, or before the selected item in both Text View and Grid View. If in Grid View the current selection is an attribute, the XInclude element is inserted after the attribute and before the first child element of the attribute's parent element. Selecting this command pops up the XInclude dialog (*screenshot below*).



The XML file to be included is entered in the `href` text box (alternatively, you can browse for the file by clicking the **Browse (...)** button to the right of the text box). The filename will be entered in the XML document as the value of the `href` attribute. The `parse`, `xpointer`, and `encoding` attributes of the XInclude element (`xi:include`), and the `fallback` child element of `xi:include` can also be inserted via the dialog. Do this by first checking the appropriate check box and then selecting/entering the required values. In the case of the `fallback` element, checking its check box only inserts the empty element. The content of the `fallback` element must be added subsequently in one of the editing views.

The `parse` attribute determines whether the included document is to be parsed as XML or text. (XML is the default value and therefore need not be specified.) The `xpointer` attribute identifies a specific fragment of the document located with the `href` attribute; it is this fragment that will be included. The `encoding` attribute specifies the encoding of the included document so that XMLSpy can transcode this document (or the part of it to be included) into the encoding of the including document. The contents of the `fallback` child element replace the `xi:include` element if the document to be included cannot be located.

Here is an example of an XML document that uses XInclude to include two XML documents:

```
<?xml version="1.0" encoding="UTF-16"?>
<AddressBook xsi:schemaLocation="http://www.altova.com/sv/myaddresses
AddressBook.xsd"
  xmlns="http://www.altova.com/stylevision/tutorials/myaddresses"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xi="http://www.w3.org/2001/XInclude">
  <xi:include href="BusinessAddressBook.xml"/>
  <xi:include href="PersonalAddressBook.xml"/>
</AddressBook>
```

When this XML document is parsed, it will replace the two XInclude elements with the files specified in the respective `href` attributes.

xml: base

When the XML validator of XMLSpy reads an XML document and encounters the `include` element in the XInclude namespace (hereafter `xi:include`), it replaces this element (`xi:include`) with the XML document named in the `href` attribute of the `xi:include` element. The document element (root element) of the included XML document (or the element identified by an XPointer) will be included with an attribute of `xml:base` in order to preserve the base URIs of the included element. If the resulting XML document (containing the included XML document/s or tree fragment/s) must be valid according to a schema, then the document element of the included document (or the top-level element of the tree fragment) must be created with a content model that allows an attribute of `xml:base`. If, according to the schema, the `xml:base` attribute is not allowed on this element, then the resulting document will be invalid. How to define an `xml:base` attribute in an element's content model using XMLSpy's Schema View is described in the [xml: Prefixed Attributes](#) section of the Schema View section of the documentation.

XPointers

XMLSpy supports XPointers in XInclude. The relevant W3C recommendations are the [XPointer Framework](#) and [XPointer element\(\) Scheme](#) recommendations. The use of an XPointer in an XInclude element enables a specific part of the XML document to be included, instead of the entire XML document. XPointers are used within an XInclude element as follows:

```
<xi:include href="PersonalAddressBook.xml" xpointer="element( usa)"/>
<xi:include href="BusinessAddressBook.xml" xpointer="element( /1/1)"/>
<xi:include href="BobsAddressBook.xml" xpointer="element( usa/3/1)"/>
<xi:include href="PatsAddressBook.xml" xpointer="
element( usa) element( /1/1)"/>
```

In the `element()` scheme of XPointer, an NCName or a child sequence directed by integers may be used.

- In the first `xi:include` element listed above, the `xpointer` attribute uses the `element` scheme with an NCName of `usa`. According to the XPointer Framework, this NCName identifies the element that has an ID of `usa`.
- In the second `xi:include` listed above, the `xpointer` attribute with a value of `element`

(/1/1) identifies, in the first step, the first child element of the document root (which, if the document is well-formed, will be its document (or root) element). In the second step, the first child element of the element located in the previous step is located; in our example, this would be the first child element of the document element.

- The `xpointer` attribute of the third `xi:include` listed above uses a combination of NCName and child sequence. This XPointer locates the first child element of the third child element of the element having an ID of `usa`.
- If you are not sure whether your first XPointer will work, you can back it up with a second one as shown in the fourth `xi:include` listed above:

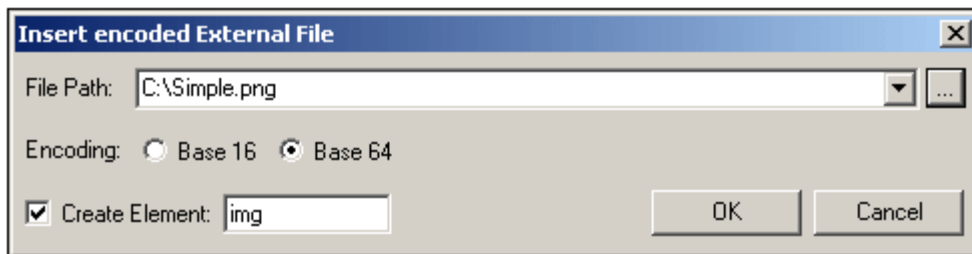
`xpointer="element(usa)element(/1/1) "`. Here, if there is no element with an ID of `usa`, the back-up XPointer specifies that the first child element of the document element is to be selected. Additional backups are also allowed. Individual XPointers may not be separated, or they may be separated by whitespace: for example,
`xpointer="element(usa)element(addresses/1) element(/1/1) "`.

Note: The namespace binding context is not used in the `element()` scheme because the `element()` scheme does not support qualified names.

Insert Encoded External File

The **Encoded External File** command is available in Text View and Grid View. It enables an external file to be included as encoded Base-16 or Base-64 text at any location in the XML document. This feature enables external files to be embedded in the XML document.

Clicking the **Insert | Encoded External File** command pops up the Insert Encoded External File dialog (*screenshot below*).



You can browse for or enter the name of the external file to be encoded and embedded. Either a Base-16 or Base-64 encoding must be specified. If you wish to enclose the encoded text in an element, then check the Create Element check box and specify the name of the desired element in the Create Element text box. If the Create Element check box is not checked, then the encoded text will be inserted directly at the cursor location.

On clicking **OK**, the encoded text of the selected file is inserted at the cursor location, with an enclosing element if this has been specified.

```
<img ext="png" encoding="xs:base64Binary">
iVBORw0KGgoAAAANSUhEUgAAABAAAAAQMAAAAPw0iAAAAB1BMVEUAAAD/
//+12Z/dAAAAM0lEQVR4nGP4/5/h/1+G/58ZDrAz3D/MCH8yw83NDDenGe4U
g9C9zwz3gVLMdA/A6P9/AFGGFyjOXZtQAAAAAE1FTkSuQmCC
</img>
```

The listing above shows the encoded text of a PNG image file. An `img` element was created around the encoded text.

21.2.8 Pretty-Print XML Text



The **Pretty-Print XML Text** command reformats your XML document in Text View. Two formatting options are available, depending upon whether the *Use Indentation* check box in the [View tab of the Options dialog](#) (**Tools | Options**) is checked or not:

- *Use Indentation* checked: The document is reformatted to give a structured display, indenting each deeper level in the hierarchy by an additional amount of the specified indentation space. This enables a clearer view of the document structure.
- *Use Indentation* unchecked: The document is reformatted so that each new line is left-aligned.

To set up a structured, indented view of the XML document, do the following:

1. In the [View tab of the Options dialog](#) (**Tools | Options**), check the *Use Indentation* check box.
2. In the [Text View Settings dialog](#) (**View | Text View Settings**), set the tab size you want for the indentation of the pretty-printed text.
3. In the [File tab of the Options dialog](#) (**Tools | Options**), enter the elements for which no output formatting (indentation) is wanted.
4. Click the **Pretty-Print XML Text** command (this command).

To reformat the document so that all lines are left-aligned, uncheck the *Use Indentation* check box.

Note the following points:

1. The XML document must be well-formed for this command to work.
2. Pretty-printing adds spaces or tabs to the document when the document is saved.
3. If pretty-printing has been switched on (**Tools | Options | View | Use Indentation**) and if you change from Text View to Grid View and back to Text View, then the document will be pretty-printed automatically. There is no need to select the Pretty-Print XML Text command.

21.2.9 Select All

Ctrl+A

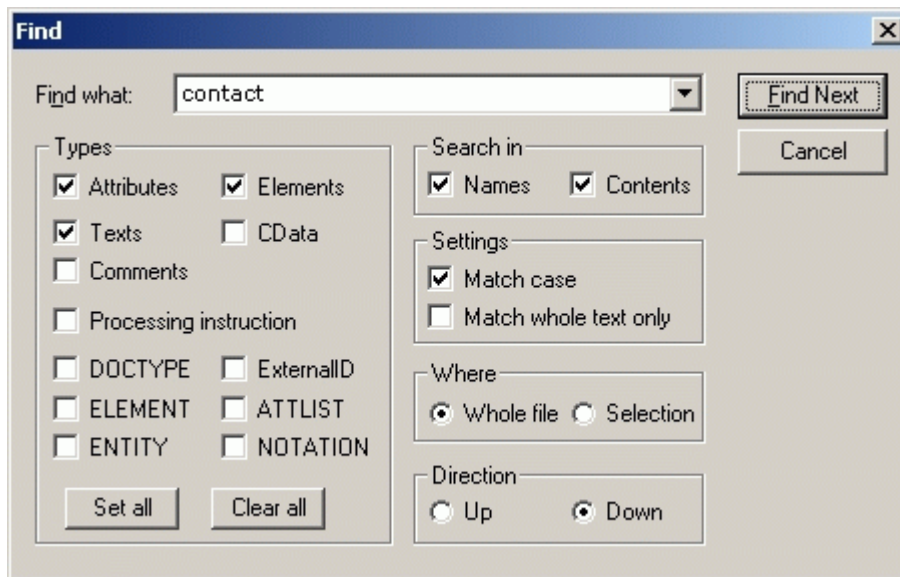
The **Select All** command selects the contents of the entire document.

21.2.10 Find, Find Next

The **Find** command (**Ctrl+F**)  pops up the Find dialog, in which you can specify the string you want to find and other options for the search. Depending on the view you are using, the Find dialog displays different options. To find text, enter the text in the Find What text box or use the combo box to select from one of the last 10 search criteria, and then specify the options for the search.

Grid View

In Grid View, the following dialog box appears. The options available are described below.

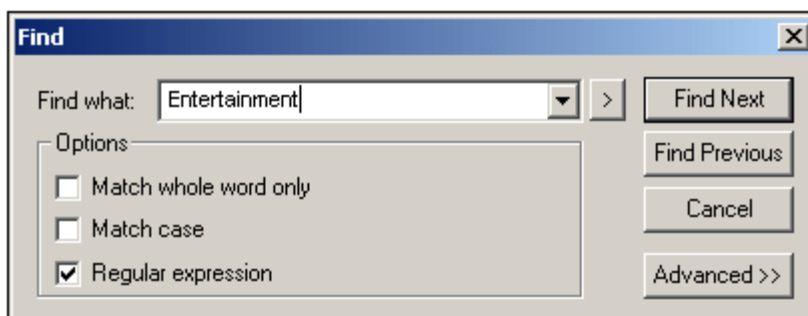


Select the options you require or select a radio button.

- The Types pane allows you to select what XML document nodes or components you wish to include in the search. This enables you to skip particular node types. The Set All button activates all the type check boxes; the Clear All button deactivates all the type check boxes.
- The Search In pane allows you to define whether the names of a node, the contents of a node, or both should be searched for the input text string.
- The Settings pane enables you to define whether the search should be case-sensitive and/or match the entire input string.
- The Where pane allows you to define the scope of the search.
- The Direction option specifies the search direction.

Text View

In Text View, the following dialog box appears. The options available are described below.

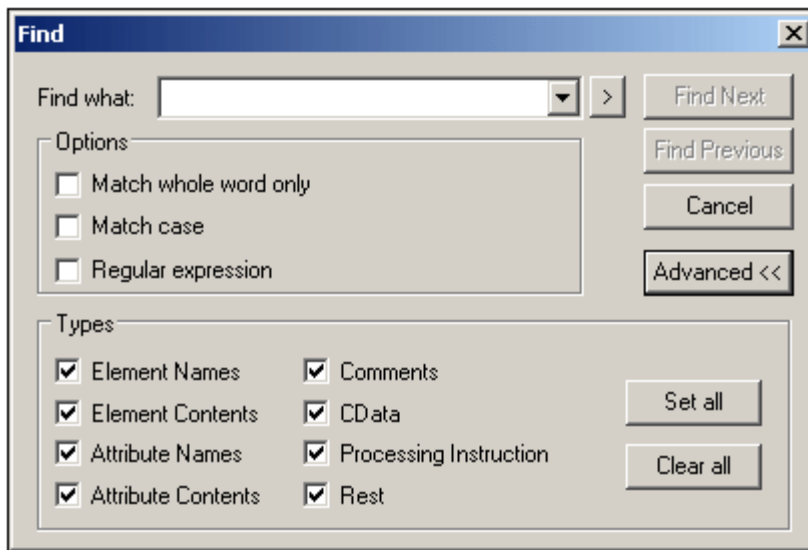


The following Find options are available:

- Match whole word only: Only the exact words in the text will be matched. For example, for the find string `fit`, with Match Whole Word checked, only the word `fit` will match the find string; the `fit` in `fitness`, for example, would not.
- Match case: Case-sensitive search (Address is not the same as address).
- Regular expression: Searches for text specified by the regular expression you enter in the text box. See [Regular expressions](#) for a description of regular expressions.

Clicking the **Advanced** button opens the Types pane (*screenshot below*), in which you can

select the type of node to search.

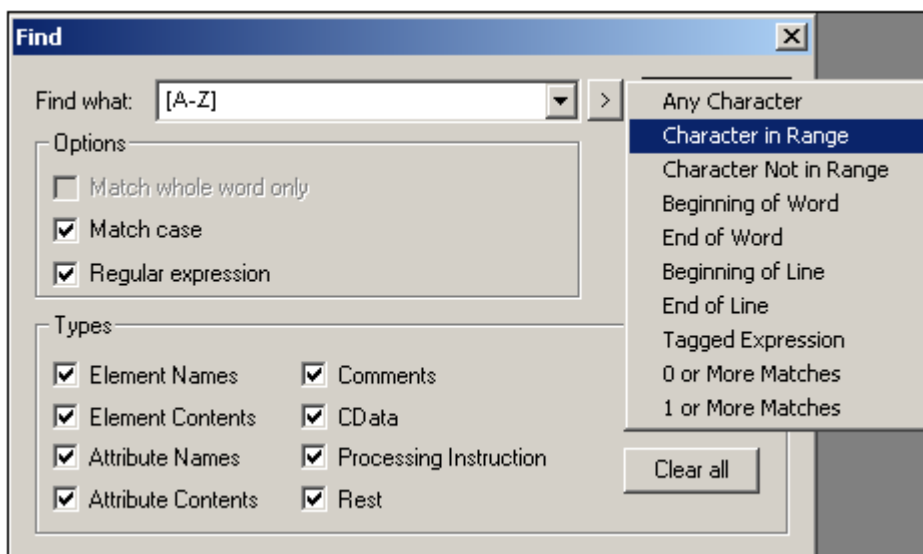


Please note:

- The Find dialog is modeless, which means that it can remain open while you continue to use Text View. Pressing Enter while the dialog box is open, closes the dialog box. If text is marked prior to opening the dialog box, then the marked text is automatically inserted into the Find What text box.
- Once the Find dialog is closed, you can repeat the current search by pressing F3 for a forward search, or Shift+F3 for a backward search.
- The unfold button to the right of the Find What combo box, opens a secondary window which you can use to enter [regular expressions](#).

Regular expressions

You can use regular expressions to further refine your search criteria. A pop-up list is available to help you build regular expressions. To access this list, click the > button to the right of the input field for the search term.



Clicking on the required expression description inserts the corresponding expression syntax in

the input field. Given below is a list of regular expression syntax characters.

.	Matches any character. This is a placeholder for a single character.
\(	Marks the start of a region for tagging a match.
\)	Marks the end of a tagged region.
\n	Where n is 1 through 9 refers to the first through ninth tagged region when replacing. For example, if the search string was Fred\([1-9]\)XXX and the replace string was Sam\1YYY, when applied to Fred2XXX this would generate Sam2YYY.
\<	Matches the start of a word.
\>	Matches the end of a word.
\x	Allows you to use a character x, that would otherwise have a special meaning. For example, \[would be interpreted as [and not as the start of a character set.
[...]	Indicates a set of characters, for example, [abc] means any of the characters a, b or c. You can also use ranges, for example [a-z] for any lower case character.
[^...]	The complement of the characters in the set. For example, [^A-Za-z] means any character except an alphabetic character.
^	Matches the start of a line (unless used inside a set, see above).
\$	Matches the end of a line. Example: A+\$ to find one or more A's at end of line.
*	Matches 0 or more times. For example, Sa*m matches Sm, Sam, Saam, Saaam and so on.
+	Matches 1 or more times. For example, Sa+m matches Sam, Saam, Saaam and so on.

Note: Regular expressions are not supported in the Replace field.

The **Find Next** command (F3)  repeats the last Find command to search for the next occurrence of the requested text.

The **Find** and **Find Next** commands can also be used to find file and folder names when a project is selected in the Project window.

21.2.11 Replace

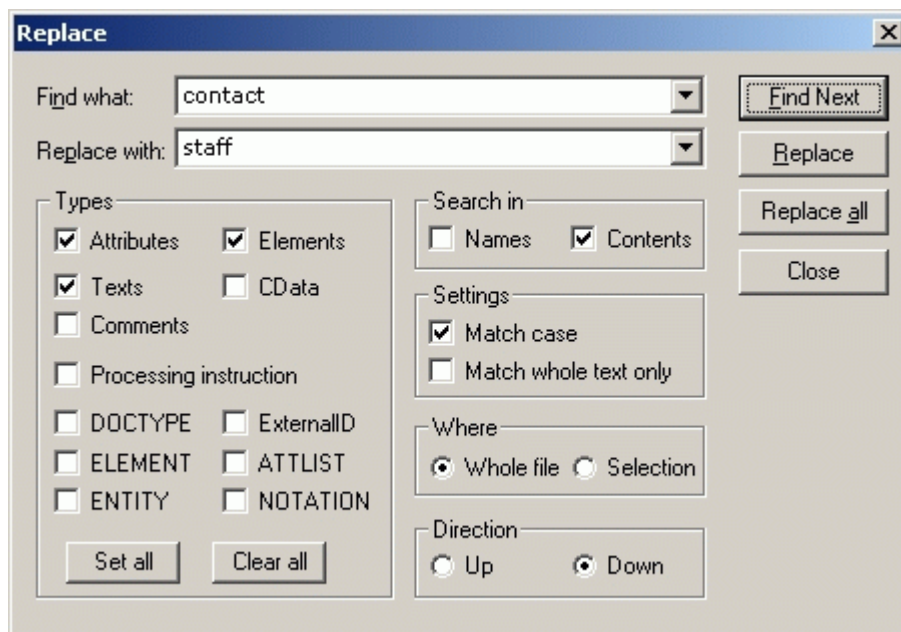


Ctrl+H

The **Replace** command enables you to find and replace one text string with another text string. It features the same options as the [Find...](#) command. Depending on the view you are using, the Replace dialog displays different find options. You can replace each item individually, or you can use the **Replace All** button to perform a global search-and-replace operation.

Grid View

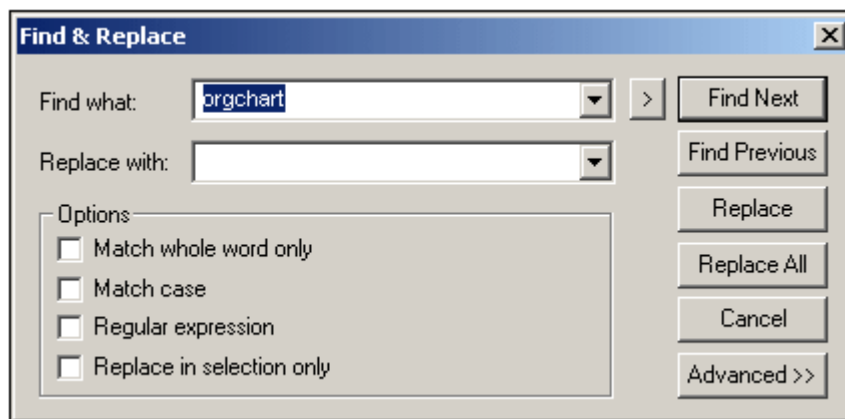
The screenshot below shows the various find options.



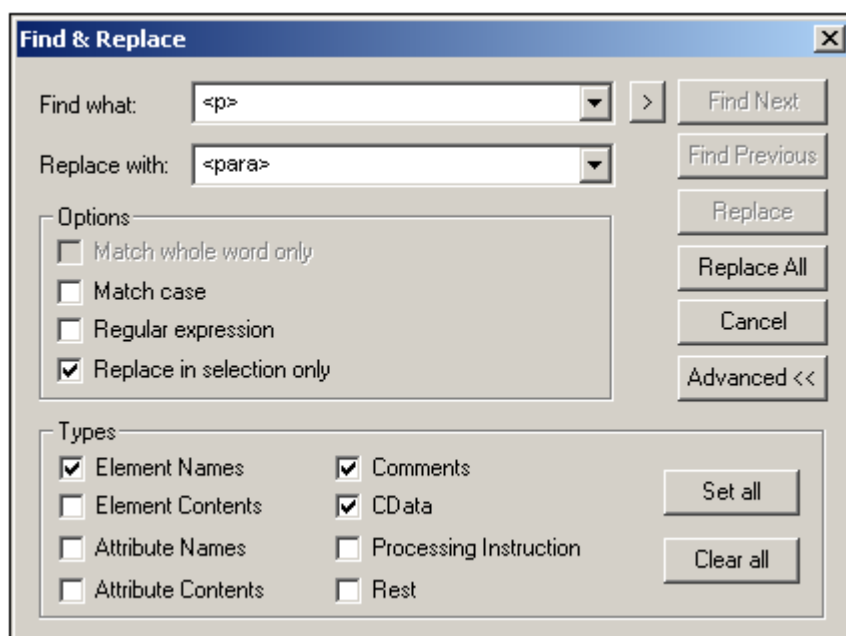
These options are described in the [Find...](#) section.

Text view

In Text View, selecting the Replace... command opens the Find & Replace dialog shown below. The options are the same as for the [Find...](#) dialog. The Replace In Selection only option carries out the find and replace operation only within the text selection.



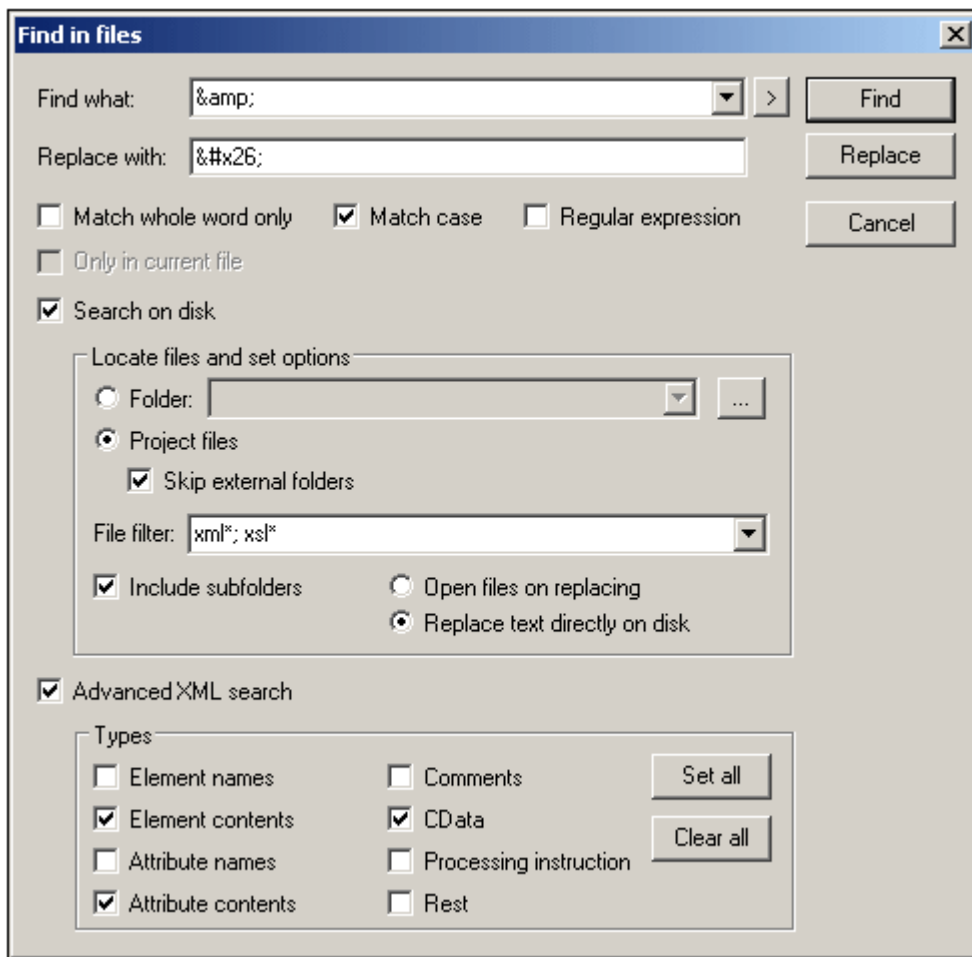
Clicking the **Advanced** button opens the Types options (see [Find...](#) for details).



Please note: When using the **Replace all** command, each replacement is recorded as a single operation, so **Replace all** can be undone step-by-step.

21.2.12 Find in Files

The **Find in Files** command is a powerful way to find and replace text quickly among a large number of files. Clicking the command pops up the Find in Files dialog (*screenshot below*). The **Find in Files** command is different from the **Find** command in that it searches all the specified locations for the Find string at once and executes replace actions at once. A report is then displayed in the [Find in Files output window](#). In the case of the **Find** command, however, the user enters the search string and goes through the (single) active document one found item at a time.



Find criteria

There are two broad find criteria: (i) what to find, and (ii) where to look? For a description of how to set the text that is to be searched (what to find), see the description of the [Find](#) command. If the text entered in the Find What text box is a regular expression, then the Regular Expression check box must be checked. An entry helper for regular expressions can be accessed by clicking the  button. The use of regular expressions for searching is explained in the section about the [Find](#) command.

To specify what node types and parts of an XML document should be searched, check the Advanced XML Search check box and then check the required node types.

You can specify what files should be searched by checking either the *Only in Current File* check box or the *Search on Disk* check box. If you choose to search on disk, you can select a folder or a [project](#) to search (after checking the Search On Disk check box). When a project folder is selected, external folders added to the project can be skipped. The files to be searched can be filtered by file extension and a star (`xml*` or `xsl*`, for example). The separator between two file extensions can be a comma or a semi-colon (`xml*; xsl*`, for example). The star character can also be used as a wildcard.

The instances of the Find string at all the search locations are listed in the [Find in Files output bar](#). Clicking on one of the listed items opens that file in Text View and highlights the item.

Replace

You should note that clicking the **Replace** button replaces all the instances of the Find string with the Replace string. If Open Files On Replacing was checked in the Find in Files dialog, then the file will be opened in Text View; otherwise the replacement is done silently. All the replaced strings are listed in the [Find in Files output bar](#). Clicking on one of the listed items opens that file in Text View and highlights the item.

Note: Regular expressions are not supported in the Replace field.

21.2.13 Bookmark Commands

Insert/Remove Bookmark

The **Insert/Remove Bookmark** command (**Ctrl+F2**)  inserts a bookmark at the current cursor position, or removes the bookmark if the cursor is in a line that has been bookmarked previously. This command is only available in Text View.

Bookmarked lines are displayed in one of the following ways:

- If the bookmarks margin has been enabled, then a solid blue ellipse appears to the left of the text in the bookmark margin.
- If the bookmarks margin has not been enabled, then the complete line containing the cursor is highlighted.

The **F2** key cycles through all the bookmarks in the document.

Remove All Bookmarks

The **Remove All Bookmarks** command (**Ctrl+Shift+F2**)  removes all the currently defined bookmarks. This command is only available in Text View. Note that the **Undo** command does not undo the effects of this command.

Goto Next Bookmark

The **Goto Next Bookmark** command (**F2**)  places the text cursor at the beginning of the next bookmarked line. This command is only available in the Text View.

Goto Previous Bookmark

The **Goto Previous Bookmark** command (**Shift+F2**)  places the text cursor at the beginning of the previous bookmarked line. This command is only available in the Text View.

21.2.14 Comment In/Out

The **Comment In/Out** command is available in Text View and is used to comment and uncomment XML text fragments. Text in an XML document can be commented out using the XML start-comment and end-comment delimiters, respectively `<!--` and `-->`. In XMLSpy, these comment delimiters can be easily inserted using the **Comment In/Out** menu command.

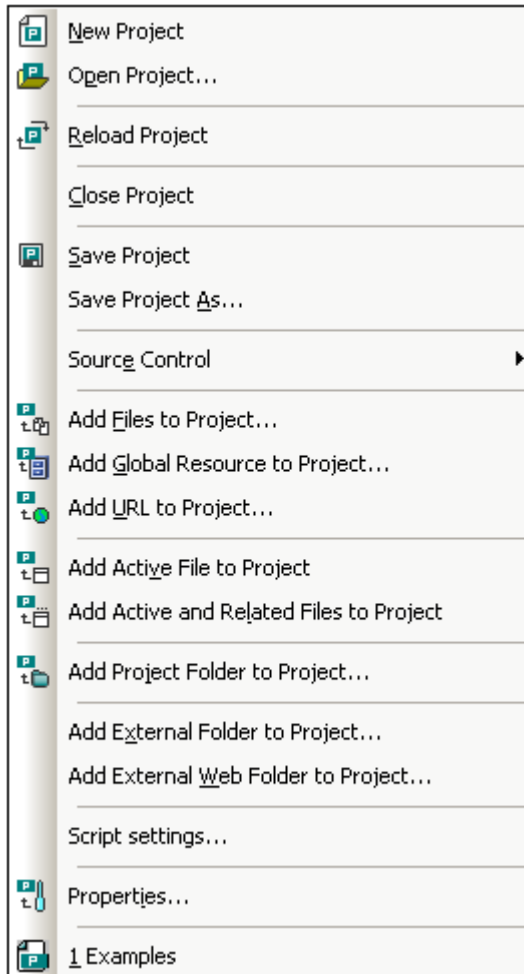
To comment out a block of text, select the text to be commented out and then select the command **Comment In/Out**, either from the **Edit** menu or the context menu that you get on right-clicking the selected text. The commented text will be grayed out (*see screenshot below*).

```
<Department>
  <Name>Administration</Name>
  <Person>
  <Person>
  <Person>
  <!--<Person>
    <First
    <Last></Last>
    <PhoneExt></PhoneExt>
    <EMail></EMail>
    <LeaveTotal></LeaveTotal>
    <LeaveUsed></LeaveUsed>
    <LeaveLeft></LeaveLeft>
  </Person>-->
</Department>
```

To uncomment a commented block of text, select the commented block **excluding** the comment delimiters, and select the command **Comment In/Out**, either from the **Edit** menu or the context menu that you get on right-clicking the selected text. The comment delimiters will be removed and the text will no longer be grayed out.

21.3 Project Menu

XMLSpy uses the familiar tree view to manage multiple files or URLs in XML projects. [Files](#) and [URLs](#) can be grouped into [folders](#) by common extension or any arbitrary criteria, allowing for easy structuring and batch manipulation.



Please note: Most project-related commands are also available in the context menu, which appears when you right-click any item in the project window.

Absolute and relative paths

Each project is saved as a project file, and has the `.spp` extension. These files are actually XML documents that you can edit like any regular XML File. In the project file, absolute paths are used for files/folders on the same level or higher, and relative paths for files/folders in the current folder or in sub-folders. For example, if your directory structure looks like this:

```
| -Folder1
|   |
|   | -Folder2
|   |   |
|   |   | -Folder3
|   |   |   |
|   |   |   | -Folder4
```

If your .spp file is located in `Folder3`, then references to files in `Folder1` and `Folder2` will look something like this:

```
c:\Folder1\NameOfFile.ext  
c:\Folder1\Folder2\NameOfFile.ext
```

References to files in `Folder3` and `Folder4` will look something like this:

```
.\NameOfFile.ext  
.\Folder4\NameOfFile.ext
```

If you wish to ensure that all paths will be relative, save the .spp files in the root directory of your working disk.

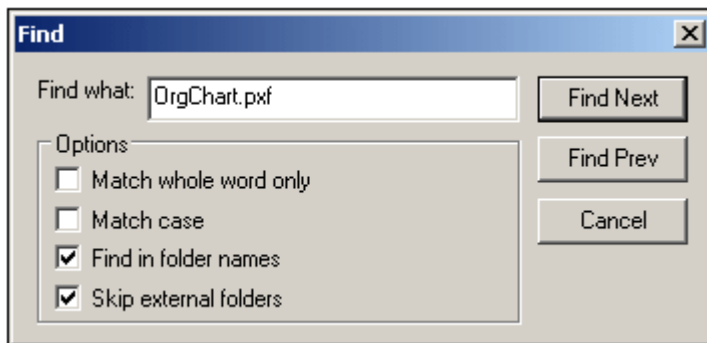
Drag-and-drop

In the Project window, a folder can be dragged to another folder or to another location within the same folder. A file can be dragged to another folder, but cannot be moved within the same folder (within which files are arranged alphabetically). Additionally, files and folders can be dragged from Windows File Explorer to the Project window.

Find in project

You can search for project files and folders using their names or a part of their name. If the search is successful, files or folders that are located are highlighted one by one.

To start a search, select the project folder in the Project sidebar that you wish to search, then select the command **Edit | Find** (or the shortcut **Ctrl+F**). In the Find dialog that pops up (*screenshot below*) enter the text string you wish to search for and select or deselect the search options (*explained below*) according to your requirements.



The following search options are available:

- Whole-word matching is more restricted since the entire string must match an entire word in the file or folder name. In file names, the parts before and after the dot (without the dot) are each treated as a word.
- It can be specified that casing in the search string must exactly match the text string in the file or folder name.
- Folder names can be included in the search. Otherwise, only file names are searched.
- [External folders](#) can be included or excluded from the search. External folders are actual folders on the system or network, as opposed to project folders, which are created within the project and not on the system.

If the search is successful, the first matching item is highlighted in the Project sidebar. You can then browse through all the returned matching items by clicking the **Find Next** and **Find Prev**

buttons in the Find dialog.

Refreshing projects

If a change is made to an external folder, this change will not be reflected in the Project Window till the project is refreshed.

Global resources in the context menu

When you right-click a folder in the Project window, in the context menu that appears, you can select the **Add Global Resource** menu item to add a [global resource](#). The menu command itself pops up the Choose Global Resource dialog, which lists all the file-type and folder-type global resources in the currently active Global Resources XML File. Select the required global resource, and it will be added to the selected project folder.

Projects and source control providers

If you intend to add an XMLSpy project to a source control repository, please ensure that the project files position in the hierarchical file system structure is one which enables you to add files only from below it (taking the root directory to be the top of the directory tree).

In other words, the directory where the **project file** is located, essentially represents the **root directory** of the project within the source control repository. Files added from above it (the project root directory) will be added to the XMLSpy project, but their location in the repository may be an unexpected one—if they are allowed to be placed there at all.

For example, given the directory structure show above, if a project file is saved in `Folder3` and placed under source control:

- Files added to Folder1 may not be placed under source control,
- Files added to Folder2 are added to the root directory of the repository, instead of to the project folder, but are still under source control,
- Files located in Folder3 and Folder4 work as expected, and are placed under source control.

21.3.1 New Project



The **New Project** command creates a **new** project in XMLSpy. If you are currently working with another project, a prompt appears asking if you want to close all documents belonging to the current project.

21.3.2 Open Project



The **Open Project...** command opens an existing project in XMLSpy. If you are currently working with another project, the previous project is closed first.

21.3.3 Reload Project



The **Reload Project** command reloads the current project from disk. If you are working in a multi-user environment, it can sometimes become necessary to reload the project from disk, because other users might have made changes to the project.

Please note: Project files (.spp files) are actually XML documents that you can edit like any regular XML File.

21.3.4 Close Project

The **Close Project** command **closes** the active project. If the project has been modified, you will be asked whether you want to save the project first. When a project is modified in any way, an asterisk is added to the project name in the Project Window.

21.3.5 Save Project, Save Project As



The **Save Project** command **saves** the current project. You can also save a project by making the project window active and clicking the  icon.

The **Save Project As** command **saves** the current project with a new name that you can enter when prompted for one.

21.3.6 Source Control

XMLSpy supports Microsoft SourceSafe and other compatible repositories. The Source Control Systems supported by XMLSpy are listed in the section [Supported Source Control Systems](#). How to install these systems is described in the section, [Installing Source Control Systems](#). This section describes the commands in the **Project | Source Control** submenu, which are used to work with the Source Control System from within XMLSpy.

Overview of the Source Control feature

The mechanism for placing files in a XMLSpy project under source control is as follows:

1. In XMLSpy, an application project folder containing the files to be placed under source control is created. Typically, the application project folder will correspond to a local folder in which the project files are located. The path to the local folder is referred to as the local path.
2. In the source control system's database (also referred to as source control or repository), a folder is created that will contain the files to be placed under source control.
3. Application project files are added to source control using the command [Project | Source Control | Add to Source Control](#).
4. Source control actions, such as checking in to, checking out from, and removing files from source control, can be carried out by using the commands in the [Project | Source Control submenu](#). The commands in this submenu are listed in the sub-sections of this

section.

Note: If you wish to change the current source control provider, this can be done in any of two ways: (i) via the Source Control options ([Tools | Options | Source Control](#)), or (ii) in the Change Source Control dialog ([Project | Source Control | Change Source Control](#)).

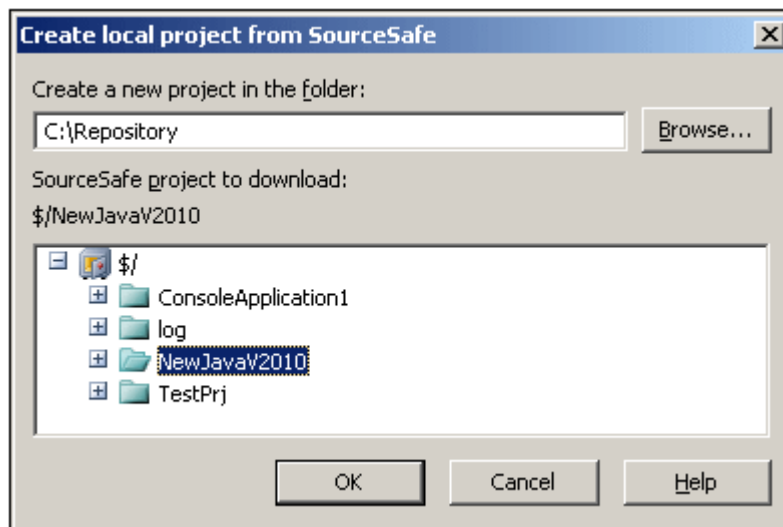
Note: Note that a Source Control project is not the same as an XMLSpy project. Source Control projects are directory-dependent, while XMLSpy projects are logical constructions without direct directory dependence.

For additional information, see the section, [Source Control](#).

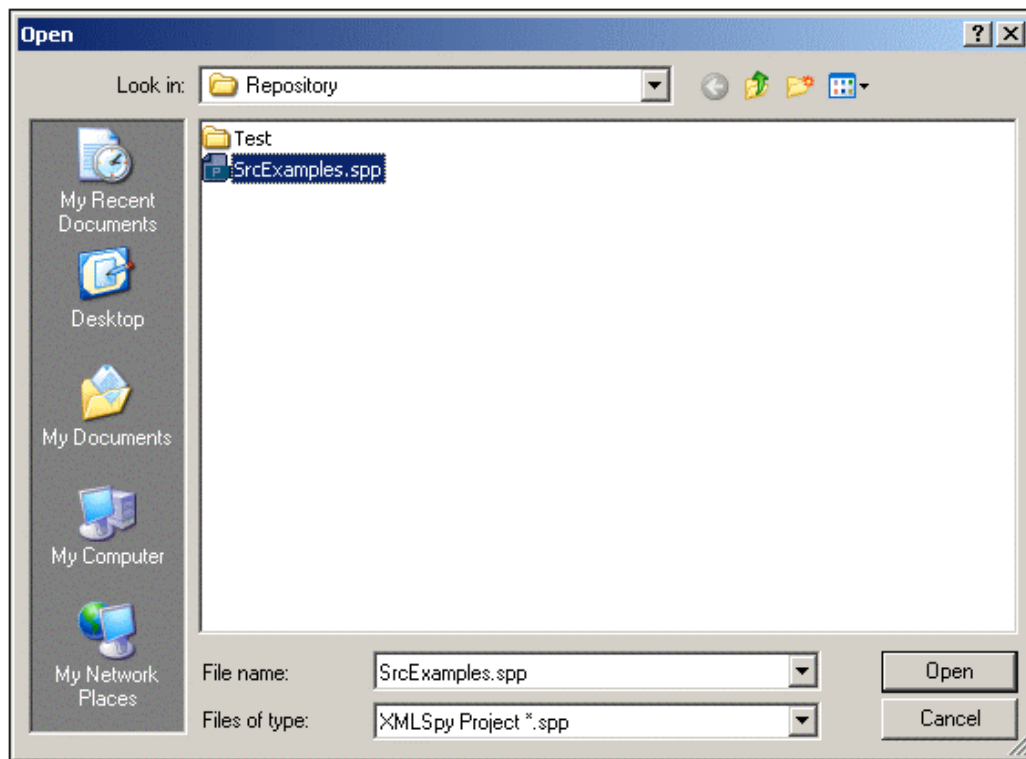
Open from Source Control

The **Open from Source Control** command creates an existing source control project locally. This command is enabled only if an XMLSpy project has previously been added to source control using the menu option [Add to Source Control](#).

1. Select **Project | Source Control | Open from Source Control**. Enter your login details in the Login dialog that appears.
2. The Create Local Project from SourceSafe dialog box appears (*screenshot below*). Select the folder to contain the local project. This folder becomes the Working Folder or Checkout Folder (in the screenshot below: C:\Repository).



3. Select the source control project you want to download. In the screenshot above, the source control project folder, `NewJavaV2010`, has been selected. Files in the source control project folder will be downloaded to the local folder. If the local folder you have specified does not exist, a dialog box opens prompting you to create it. Click **Yes** to confirm the new folder.
4. The Open dialog now pops up (*screenshot below*).



5. Click the application project file you wish to create and click **Open**. The selected application project will open in XMLSpy, and the file is placed under source control.

Source control symbols

Folders and files display certain symbols, the meanings of which are given 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.

Enable Source Control

The **Enable Source Control** command allows you to enable or disable source control for a XMLSpy project. Selecting this option on any file or folder, enables/disables source control for the whole project.

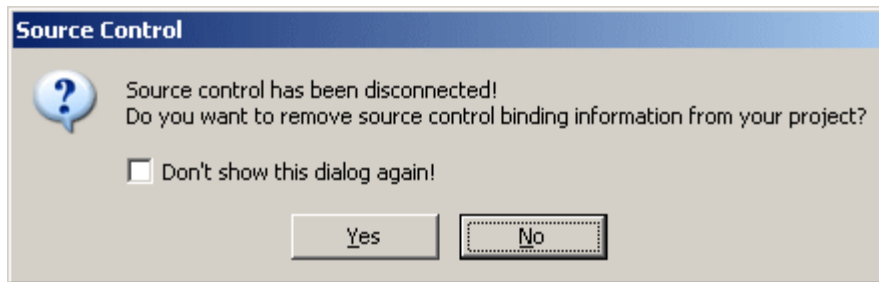
Enabling Source Control for a project

To enable source control for a project, do the following:

1. Click on any file or folder in the Project window.
2. Select the menu option **Project | Source Control | Enable Source Code Control**. The previous check in/out status of the various files are retrieved and displayed in the Project window.

Disabling Source Control for a project

To disable source control for a project, select the menu option **Project | Source Control | Enable Source Control**. You are now prompted if you want to remove the binding information from the project (see *screenshot below*).



To provisionally disable source control for the project select **No**. To permanently disable source control for the project select **Yes**.

Get Latest Version

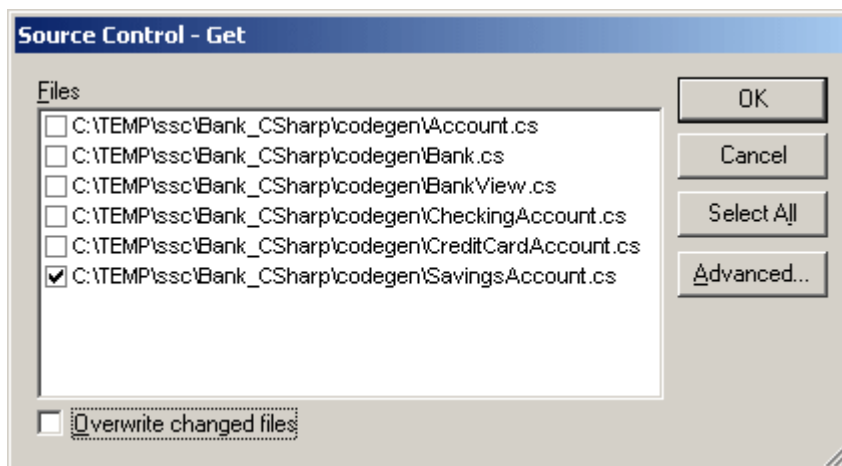
The **Get Latest Version** command retrieves and places the latest source control version of the selected file(s) in the working directory. The files are retrieved as read-only and are not checked out. No further options are available when using this command.

To get the latest version of a file, do the following:

1. Select the file(s) you want to get the latest version of in the Model Tree.
2. Select **Project | Source Control | Get Latest Version**.

Get

The **Get** command retrieves read-only copies of the selected files and places them in the working folder. By default, the files are not checked-out for editing. To get a file, select it in the Project window (multiple files can be selected) and then select the **Get** command. This pops up the Get dialog (*screenshot below*). Check the files you want to get.



Overwrite changed files check box

Overwrites those files that have been changed locally with those from the source control

database.

Select All

Selects all the files in the list box.

Advanced

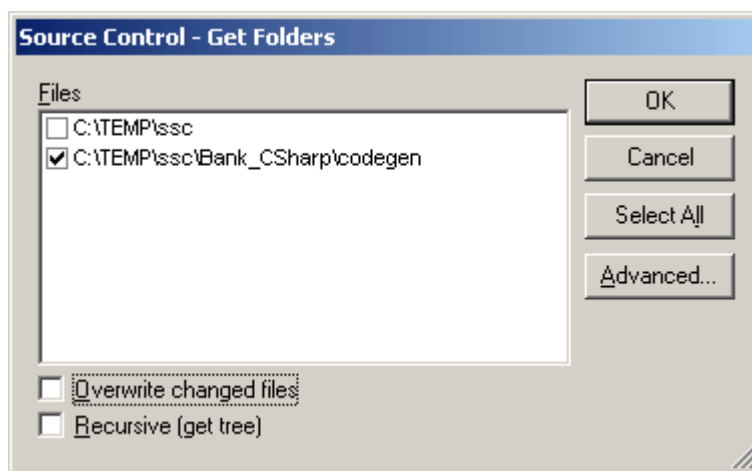
Allows you to define the *Replace writable* and *Set timestamp* options in the respective combo boxes.



The *Make writable* check box removes the read-only attribute of the retrieved files.

Get Folders

The **Get Folders** command retrieves read-only copies of files in the selected folders and places them in the working folder. By default, the files are not checked-out for editing. To get a folder, select it in the Project window and then select the **Get Folders** command. This pops up the Get Folders dialog (*screenshot below*). Check the folders you want to get.

**Overwrite changed files check box**

Overwrites those files that have been changed locally with those from the source control database.

Recursive (get tree) check box

Retrieves all files of the folder tree below the selected folder.

Select All

Selects all the files in the list box.

Advanced

Allows you to define the *Replace writable* and *Set timestamp* options in the respective combo boxes.



The *Make writable* check box removes the read-only attribute of the retrieved files.

Check Out

The **Check Out** command checks out the latest version of the selected files and places writable copies in the working directory. The files are flagged as checked out for all other users. To check out files, do the following:

1. Select the file or folder you want to check out in the Model Tree.
2. Select **Project | Source Control | Check Out**.
3. In the Check Out dialog that pops up, select the files to check out, then click **OK**.

Note the following points:

- You can change the number of files to check out by activating the individual check boxes in the Files list box.
- The *Checkout local version* option checks out only the local versions of files, not those from the source control database.
- The following items can be checked out: (i) single files (in the Project window, click the required file to select it; use **Ctrl+Click** to select multiple files, use **Ctrl+Click**); (ii) folders (in the Project window, click the required folder to select it; use **Ctrl+Click** to select multiple folders).
- Checked out files and folders are indicated with a red check mark.

Advanced

Allows you to define the *Replace writable* and *Set timestamp* options in the respective combo boxes.



The *Make writable* check box removes the read-only attribute of the retrieved files.

Source control symbols

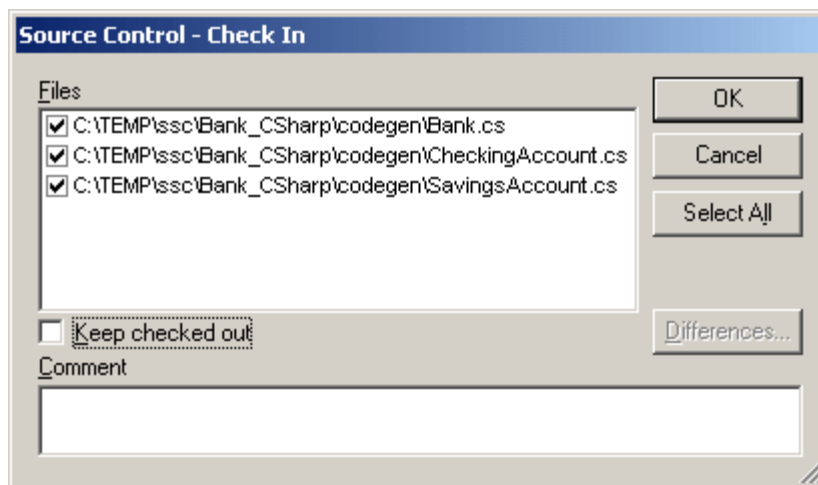
Folders and files display certain symbols, the meanings of which are given below.

	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.

Check In

The Check In command checks in previously checked out files (that is, your locally updated files) and places them in the source control database. To check in files, do the following:

1. Select the files in the Model Tree.
2. Select **Project | Source Control | Check In**.
3. In the Check In dialog that pops up, select the files to check in, then click **OK**.



Note the following points:

- As a shortcut for the Check In command, right-click a checked out item in the project window and select **Check In** from the context menu.
- The following items can be checked in: (i) single files (in the Project window, click the required file to select it; use **Ctrl+Click** to select multiple files, use **Ctrl+Click**); (ii) folders (in the Project window, click the required folder to select it; use **Ctrl+Click** to select multiple folders).
- The lock symbol denotes that the file/folder is under source control, but is currently not checked out.
- The **Differences** button is enabled when a line in the Files pane is selected. Clicking it enables you to see differences between two file versions.

Source control symbols

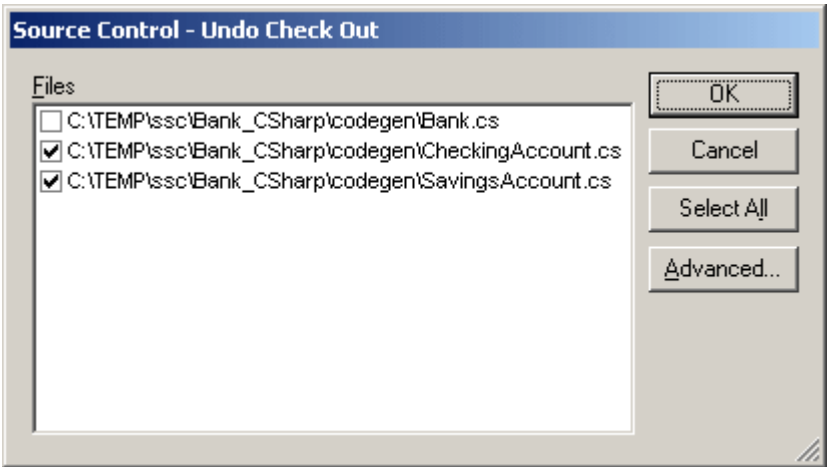
Folders and files display certain symbols, the meanings of which are given below.

	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.

Undo Check Out

The **Undo Check Out** command rejects changes made to previously checked out files (that is, your locally updated files) and retains the old files from the source control database. To undo a check out, do the following:

- 1. Select the files in the Project window.
- 2. Select **Project | Source Control | Undo Check Out**.
- 3. In the Undo Check Out dialog that pops up, select the files for which check out should be undone, then click **OK**.



Check out of the following items can be undone: (i) single files (in the Project window, click the required file to select it; use **Ctrl+Click** to select multiple files, use **Ctrl+Click**); (ii) folders (in the Project window, click the required folder to select it; use **Ctrl+Click** to select multiple folders).

Advanced

Allows you to define the *Replace writable* and *Set timestamp* options in the respective combo boxes.



The *Make writable* check box removes the read-only attribute of the retrieved files.

Source control symbols

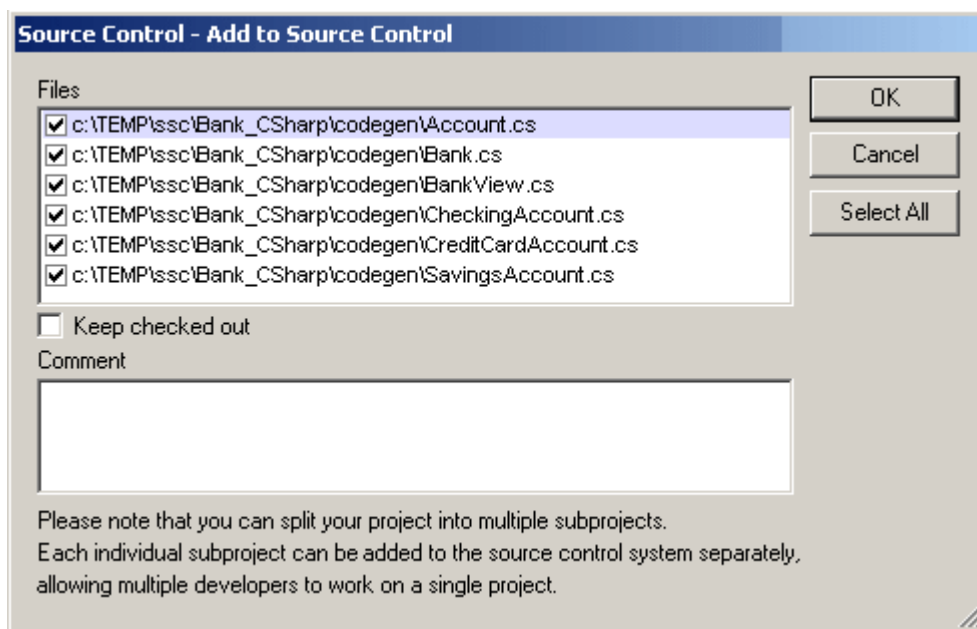
Folders and files display certain symbols, the meanings of which are given below.

	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.

Add to Source Control

The **Add to Source Control** command adds the selected files in a project folder to the source control database and places them under source control. To add files to source control, do the following:

1. In the Project window, click a file and select the menu option **Project | Source Control | Add to Source Control**. If you select a folder, the files in that folder will be added to source control.
2. In the Add to Source Control dialog that pops up, ensure that the files to be added are checked. If the file to be added should be kept checked out, check the *Keep checked out* check box. Then click **OK**.



The lock symbol now appears next to each of the files placed under source control. If the files are checked out they will be shown with a red check symbol.

Source control symbols

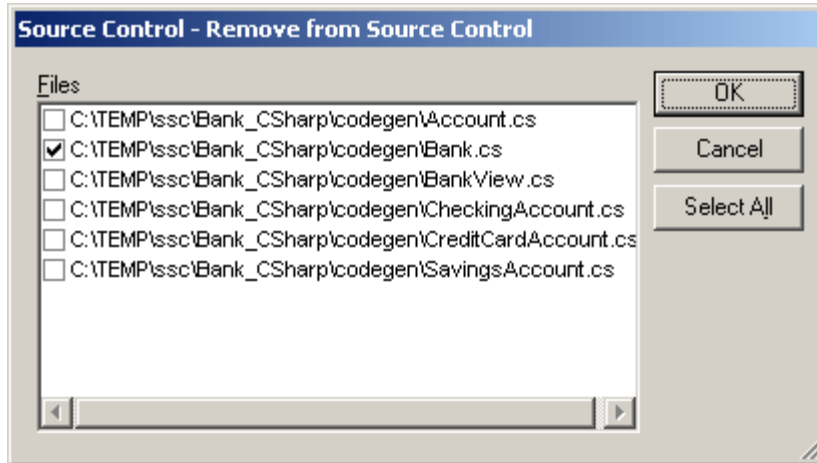
Folders and files display certain symbols, the meanings of which are given below.

	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.

Remove from Source Control

The **Remove from Source Control** command removes previously added files from the source control database. These files will be visible in the Project window but cannot be checked in or out. Use the **Add to Source Control** command to place them back under source control. To remove files from the source control provider, do the following:

1. In the Project window, select the files you wish to remove.
2. Select **Project | Source Control | Remove from Source Control**.
3. In the Remove from Source Control dialog that pops up, select the files to remove, then click **OK**.

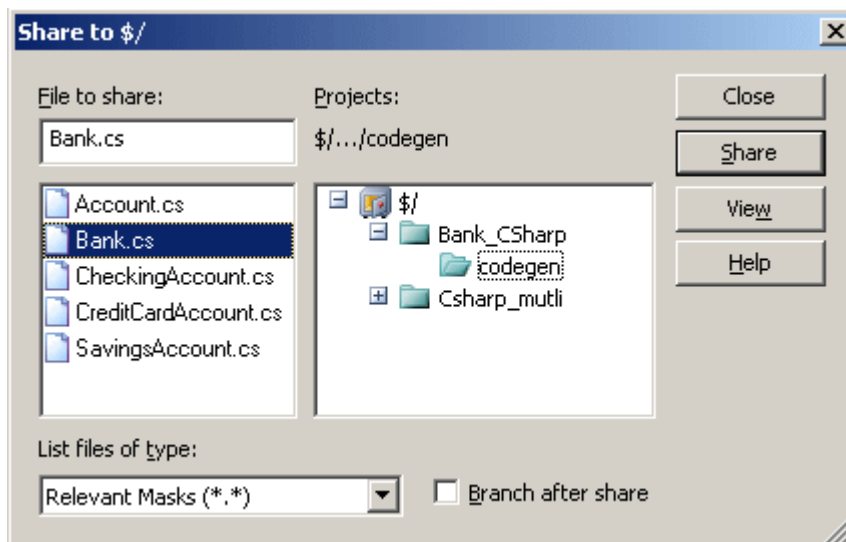


The following items can be removed from source control: (i) single files (in the Project window, click the required file to select it; use **Ctrl+Click** to select multiple files, use **Ctrl+Click**); (ii) folders (in the Project window, click the required folder to select it; use **Ctrl+Click** to select multiple folders).

Share from Source Control

To use the **Share from Source Control** command you must have the Check in/out rights to the project you are sharing from. To share a file from source control, do the following:

1. In the Project window, select the file you want to share and select **Project | Source Control | Share from Source Control**.
2. In the *Projects* list, select the project folder that contains the file you want to share.



3. In the *Files to share* list box, select the file you want to share and click the **Share** button. The file will be removed from the *Files to share* list.
4. Click the **Close** button to continue.

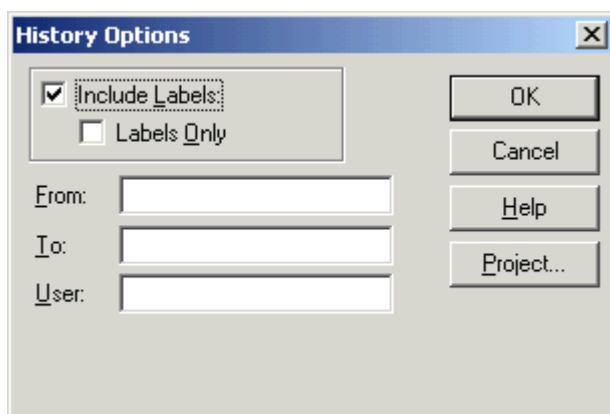
Branch after share

The *Branch After Share* option shares the file and creates a new branch to create a separate version.

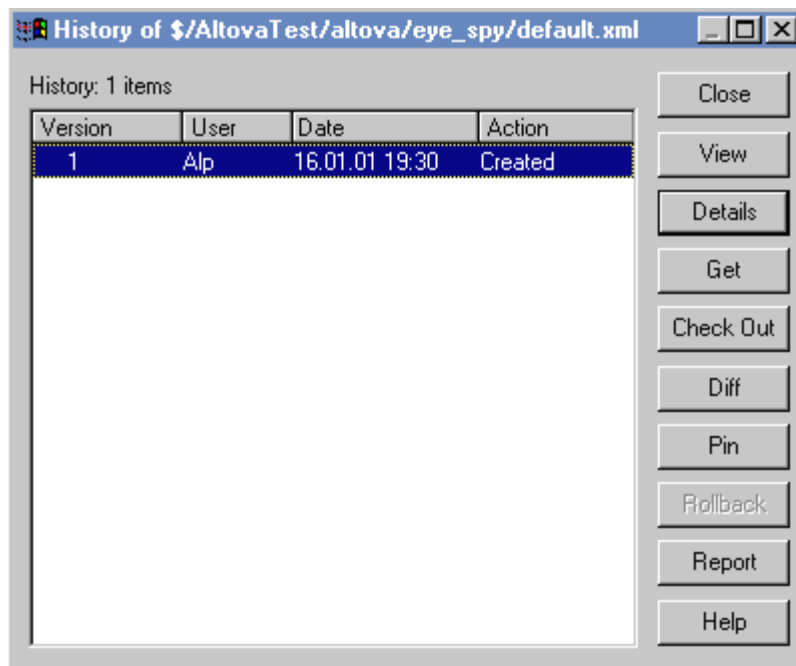
Show History

The **Show History** command displays the history of a file under source control and allows you to view detailed history info of previous versions of a file, view differences and retrieve previous versions of the file. To show the history of a file, do the following:

1. Click on the file in the Project window.
2. Select the menu option **Project | Source control | Show History**. A dialog box prompting for more information may appear (this example uses Visual Source-Safe).



3. Select the appropriate entries and confirm with **OK**.



This dialog box provides various way of comparing and getting specific versions of the file in question. Double-clicking an entry in the list opens the History Details dialog box for that file. The buttons in the dialog provide the following functionality:

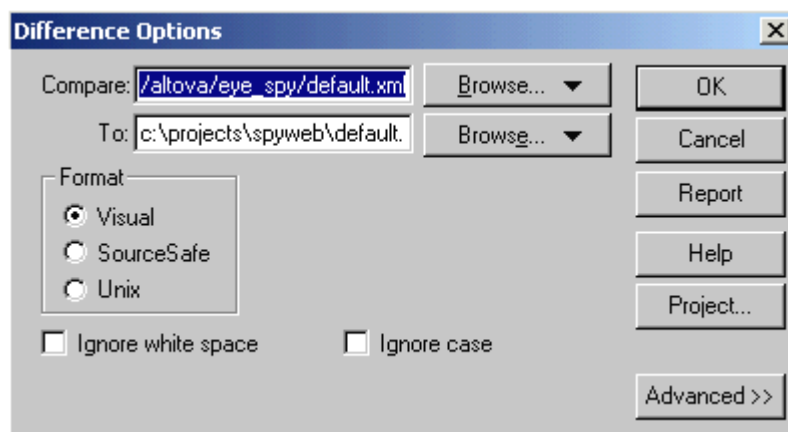
- **Close:** closes this dialog box.
- **View:** opens a further dialog box in which you can select the type of viewer with which you wish to see the file.
- **Details:** opens a dialog box in which you can see the [properties](#) of the currently active file.
- **Get:** allows you to retrieve one of the previous versions of the file in the version list and place it in the working directory.
- **Check Out:** allows you to check out a previous version of the file.
- **Diff:** opens the [Difference options](#) dialog box, which allows you to define the difference options when viewing the differences between two file versions. Use **CTRL+Click** to mark two file versions in this window, then click Diff to view the differences between them.
- **Pin:** pins or unpins a version of the file, allowing you to define the specific file version to use when differencing two files.
- **Rollback:** rolls back to the selected version of the file.
- **Report:** Generates a history report which you can send to the printer, file, or clipboard.
- **Help:** opens the online help of the source control provider plugin.

Show Differences

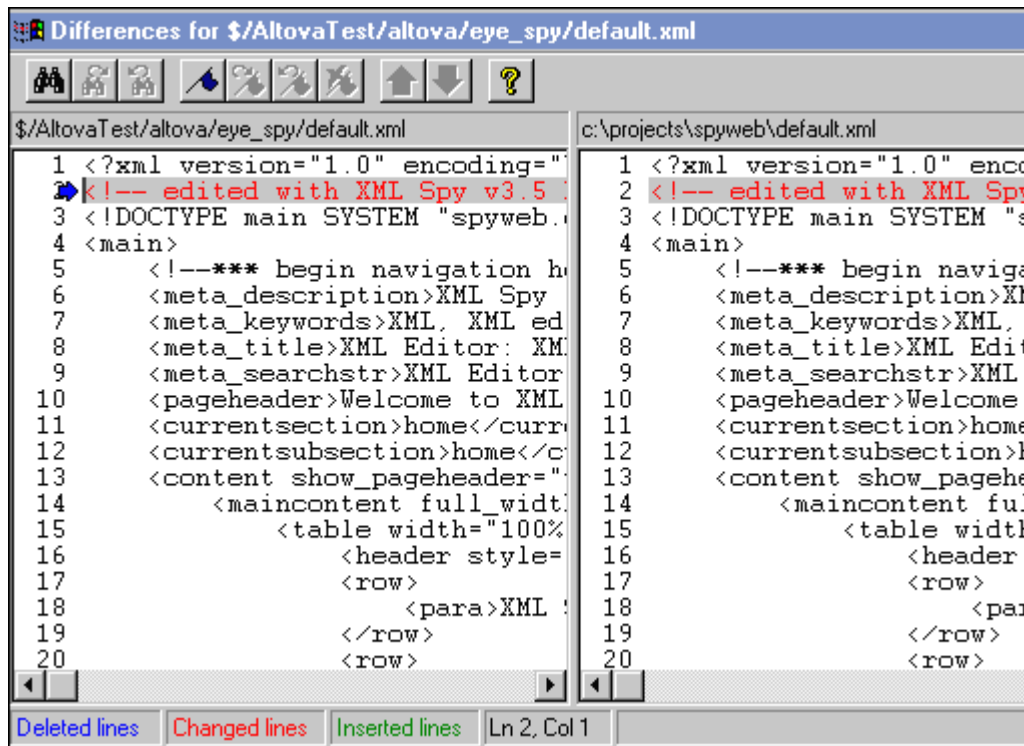
The **Show Differences** command displays the differences between the file currently in the source control repository and the **checked in/out** file of the same name in the working directory. If you have "pinned" one of the files in the history dialog box, then the pinned file will be used in the "Compare" text box. Any two files can be selected using the Browse buttons.

To show the differences between two files, do the following:

1. Check out a file from your project. Click on the file in the project window.
2. Select the menu option **Project | Source control | Show Differences**. A dialog box prompting for more information may appear at this time.



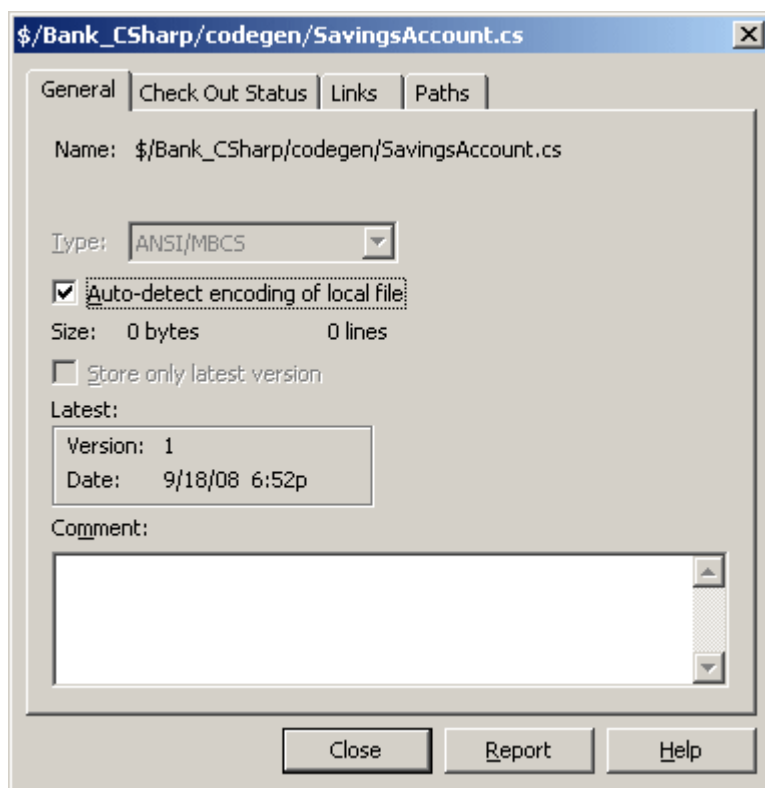
3. Select the appropriate entries and confirm with **OK**.



The differences between the two files are highlighted in both windows (this example uses MS Source-Safe).

Show Properties

The **Show Properties** command displays the properties of the currently selected file (*screenshot below*). The properties displayed depends on the source control provider you use.



Note that this command can only be used on single files.

Refresh Status

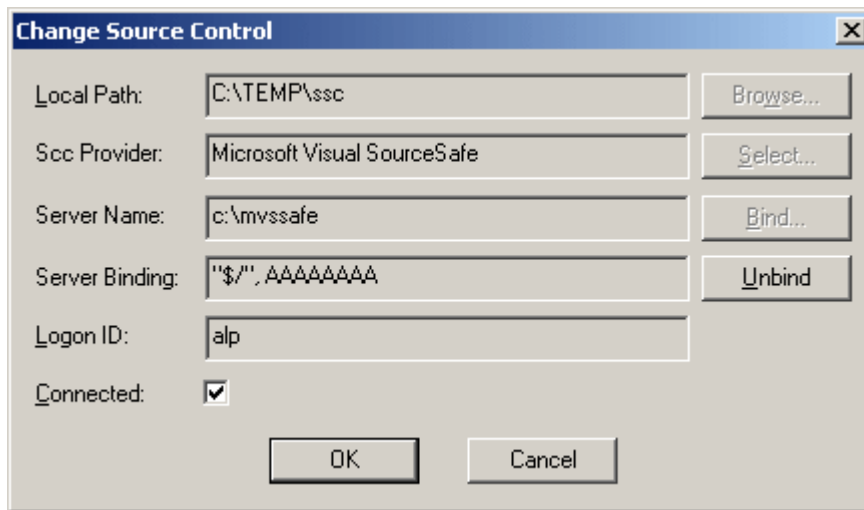
The **Refresh Status** command refreshes the status of all project files independent of their current status.

Source Control Manager

The **Source Control Manager** command starts your source control software with its native user interface.

Change Source Control

The **Change Source Control** command pops up the Change Source Control dialog box (*screenshot below*), which enables you to change the source control provider that you are using. Click the **Unbind** button first, then click the **Select** button to select the new source control provider.



After you have selected the source control provider, you will need to create a binding between the local folder and the folder on the source control server. The local folder is that entered in the Local Path text box. To select a folder on the source control server, click the **Bind** button. This enables you to log on to the server and then navigate to the required folder on the source control server. Click **OK** when done.

21.3.7 Add Files to Project



The **Project | Add Files to Project** command adds files to the current project. Use this command to add files to any folder in your project. You can either select a single file or any group of files (using **Ctrl+click**) in the Open dialog box. If you are adding files to the project, they will be distributed among the respective folders based on the File Type Extensions defined in the [Project Properties](#) dialog box.

21.3.8 Add Global Resource to Project

The **Project | Add Global Resource to Project** command pops up the Choose Global Resource dialog, in which you can select a global resource of file or folder type to add to the project. If a file-type global resource is selected, then the file is added to the appropriate folder based on the File Type Extensions defined in the [Project Properties](#) dialog box. If a folder-type global resource is selected, that folder will be opened in a file-open dialog and you will be prompted to select a file; the selected file is added to the appropriate folder based on the File Type Extensions defined in the [Project Properties](#) dialog box. For a description of global resources, see the Global Resources section in this documentation.

21.3.9 Add URL to Project



The **Project | Add URL to Project** command adds a URL to the current project. URLs in a project cause the target object of the URL to be included in the project. Whenever a batch operation is performed on a URL or on a folder that contains a URL object, XMLSpy retrieves

the document from the URL, and performs the requested operation.

21.3.10 Add Active File to Project



The **Project | Add Active File to Project** command adds the active file to the current project. If you have just opened a file from your hard disk or through an URL, you can add the file to the current project using this command.

21.3.11 Add Active And Related Files to Project



The **Project | Add Active and Related Files to Project** command adds the currently active XML document and all related files to the project. When working on an XML document that is based on a DTD or Schema, this command adds not only the XML document but also all related files (for example, the DTD and all external parsed entities to which the DTD refers) to the current project.

Please note: Files referenced by processing instructions (such as XSLT files) are not considered to be related files.

21.3.12 Add Project Folder to Project



The **Project | Add Project Folder to Project** command adds a new folder to the current project. Use this command to add a new folder to the current project or a sub-folder to a project folder. You can also access this command from the context-menu when you right-click on a folder in the project window.

Note: A project folder can be dragged and dropped into another project folder or to any other location in the project. Also, a folder can be dragged from Windows (File) Explorer and dropped into any project folder.

Note: Project folders are green, while [external folders](#) are yellow.

21.3.13 Add External Folder to Project

The **Project | Add External Folder to Project** command adds a new external folder to the current project. Use this command to add a local or network folder to the current project. You can also access this command from the context-menu when you right-click a folder in the project window.

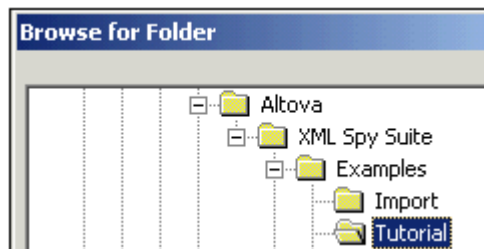
Note: External folders are yellow, while [project folders](#) are green.

Note: Files contained in external folders cannot be placed under source control.

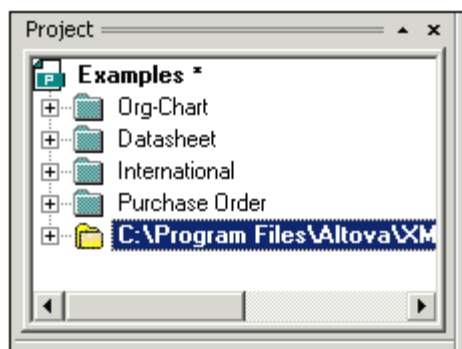
Adding external folders to projects

To add an external folder to the project:

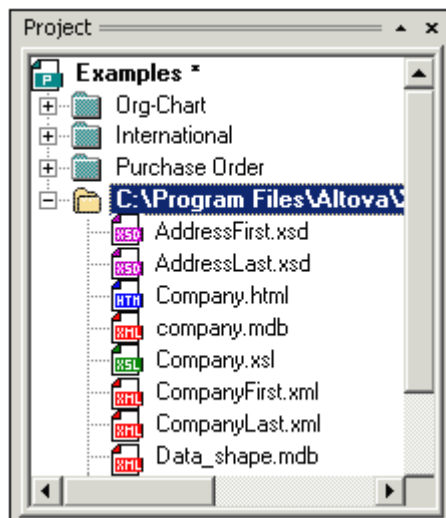
1. Select the menu option **Project | Add External Folder to Project**.
2. Select the folder you want to include from the Browse for Folder dialog box, and click **OK** to confirm.



The selected folder now appears in the project window.



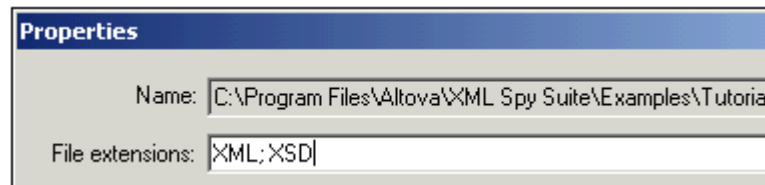
3. Click the plus icon to view the folder contents.



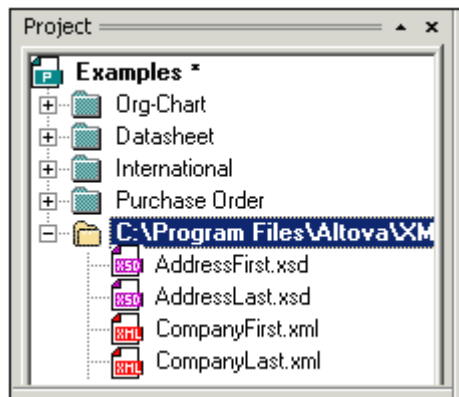
Filtering contents of folders

To filter the contents of the folder:

1. Right-click the local folder, and select the popup menu option **Properties**. This opens the Properties dialog box.



2. Click in the **File extensions** field and enter the file extensions of the file types you want to see. You can separate each file type with a **semicolon** to define multiple types (XML and Schema XSDs in this example).
3. Click **OK** to confirm.



The Project window now only shows the XML and XSD files of the tutorial folder.

Validating external folders

To validate and check an external folder for well-formedness:

1. Select the file types you want to see or check from the external folder,
2. Click the folder and click the **Check well-formedness**  or **Validate**  icon (hotkeys **F7** or **F8**). All the files visible under the folder are checked. If a file is malformed or invalid, then this file is opened in the Main Window, allowing you to edit it.
3. Correct the error and run the validation process once more to recheck.

Updating a project folder

You might add or delete files in the local or network directory at any time. To update the folder view, right-click the external folder, and select the popup menu option **Refresh external folder**.

Deleting external folders and files in them

Select an external folder and press the **Delete** key to delete the folder from the Project window. Alternatively, right-click the external folder and select the **Delete** command. Each of these actions only deletes the external folder from the Project window. The external folder is not deleted from the hard disk or network.

To delete a file in an external folder, you have to delete it physically from the hard disk or network. To see the change in the project, refresh the external folder contents (right-click the external folder and select **Refresh**).

Note: An external folder can be dragged and dropped into a project folder or to any other location in the project (but not into another external folder). Also, an external folder can be dragged from Windows (File) Explorer and dropped into any location in the project window except into another external folder.

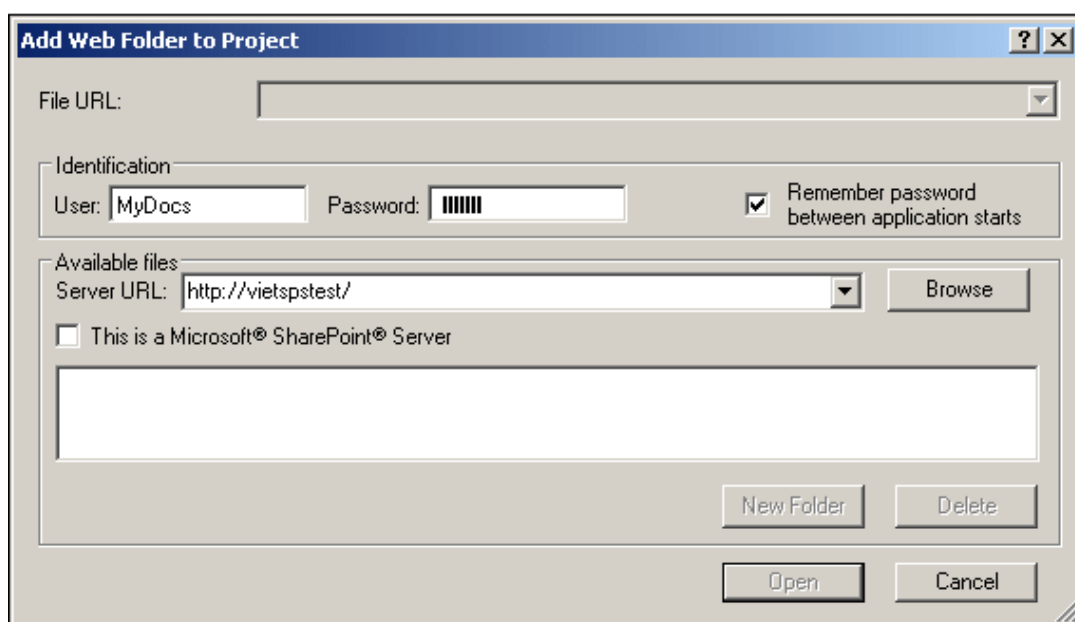
21.3.14 Add External Web Folder to Project

This command adds a new external web folder to the current project. You can also access this command from the context-menu when you right-click a folder in the project window. Note that files contained in external folders cannot be placed under source control.

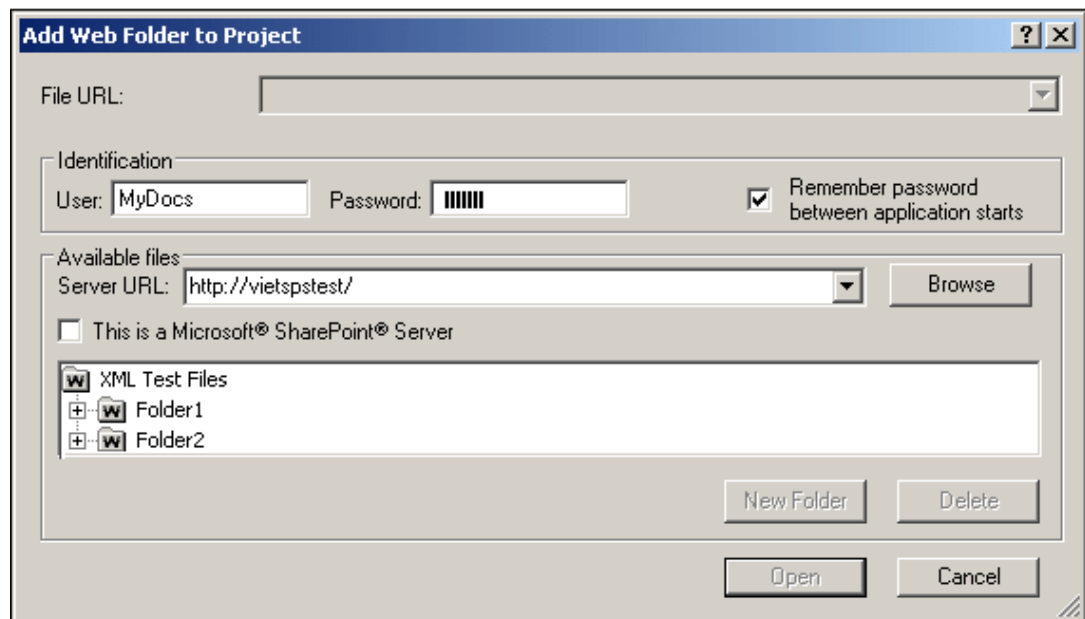
Adding an external web folder to the project

To add an external web folder to the project, do the following:

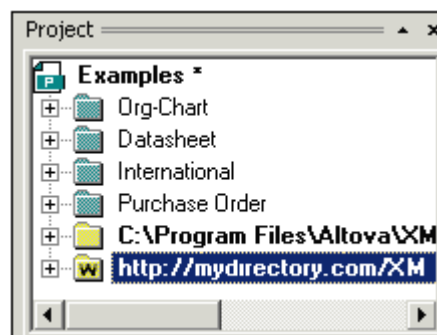
1. Select the menu option **Project | Add External Web Folder to Project**. This opens the Add Web Folder to Project dialog box (*screenshot below*).



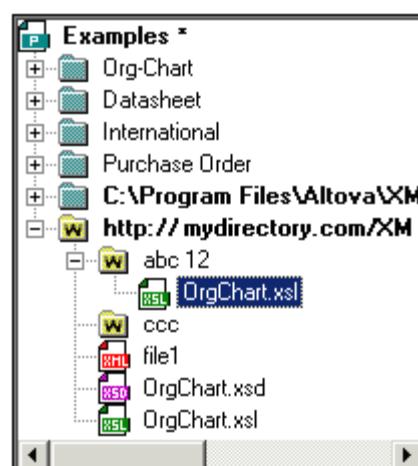
2. Click in the Server URL field and enter the URL of the server URL. If the server is a Microsoft® SharePoint® Server, check this option. See the *Folders on a Microsoft® SharePoint® Server* section below for further information about working with files on this type of server.
3. If the server is password-protected, enter your User ID and password in the *User* and *Password* fields.
4. Click **Browse** to connect to the server and view the available folders.



5. Click the folder you want to add to the project view. The **Open** button only becomes active once you do this. The URL of the folder now appears in the File URL field.
6. Click **Open** to add the folder to the project.



7. Click the plus icon to view the folder contents.



Filtering folder contents

To filter the contents of a folder, right-click the folder and select **Properties** from the context menu. In the Properties dialog that pops up, click in the *File Extensions* field and enter the file extensions of the file types you want to see (for example, XML and XSD files). Separate each file type with a semicolon (for example: `xml; xsd; sps`). The Project window will now show that folder only with files having the specified extension.

Validating and checking a folder for well-formedness

To check the files in a folder for well-formedness or to validate them, select the folder and then

click the **Check well-formedness**  or **Validate**  icon (hotkeys **F7** or **F8**, respectively). All the files that are visible in the folder are checked. If a file is malformed or invalid, then this file is opened in the main window, allowing you to edit it. Correct the error and restart the process to recheck the rest of the folder. Note that you can select discontinuous files in the folder by holding **Ctrl** and clicking the files singly. Only these files are then checked when you press **F7** or **F8**.

Updating the contents of the project folder

Files may be added or deleted from the web folder at any time. To update the folder view, right-click the external folder and select the context menu option **Refresh**.

Deleting folders and files

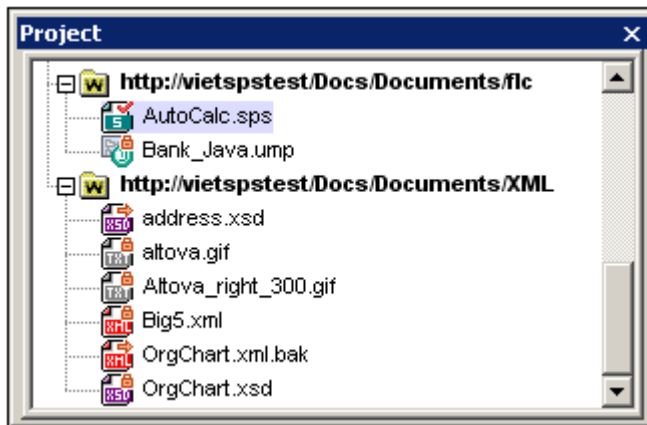
Since it is the Web folder that has been added to the project, it is only the Web folder (and not files within it) that can be deleted from the project. You can delete a Web folder from a project, by either (i) right-clicking the folder and selecting **Delete**, or (ii) selecting the folder and pressing the **Delete** key. This only deletes the folder from the Project view; it does not delete anything on the web server.

Note: Right-clicking a single file and pressing the **Delete** key does not delete a file from the Project window. You have to delete it physically on the server and then refresh the contents of the external folder.

Folders on a Microsoft® SharePoint® Server

When a folder on a Microsoft® SharePoint® Server has been added to a project, files in the folder can be checked out and checked in via commands in the context menu of the file listing in the Project window (see *screenshot below*). To access these commands, right-click the file you wish to work with and select the command you want (**Check Out**, **Check In**, **Undo Check Out**).

The User ID and password can be saved in the [properties of individual folders in the project](#), thereby enabling you to skip the verification process each time the server is accessed.

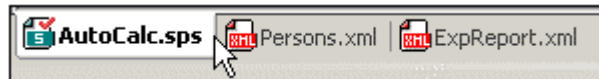


In the Project window (*screenshot below*), file icons have symbols that indicate the check-in/check-out status of files. The various file icons are shown below:

	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.

The following points should be noted:

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

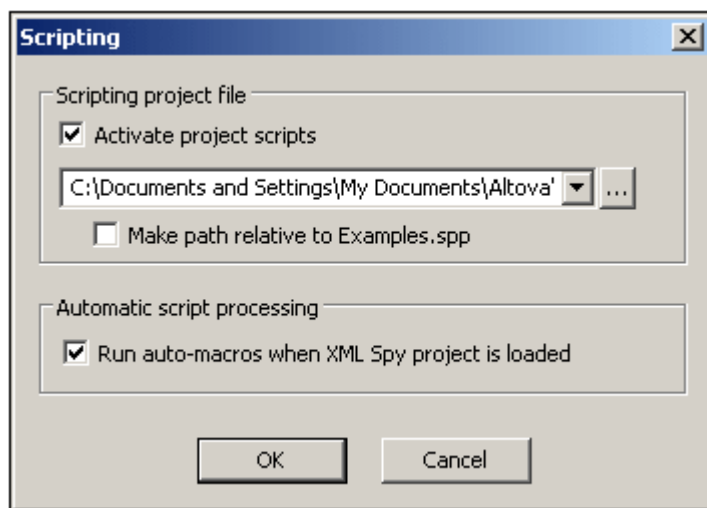


- When a file is checked out by another user, it is not available for check out.
- When a file is checked out locally by you, you can undo the check-out with the Undo Check-Out command in the context menu. This has the effect of returning the file unchanged to the server.
- If you check out a file in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

21.3.15 Script Settings

A scripting project is assigned to an XMLSpy project as follows:

1. In the XMLSpy GUI, open the required application project.
2. Select the menu command **Project | Script Settings**. The Scripting dialog (*screenshot below*) opens.



3. Check the *Activate Project Scripts* check box and select the required scripting project (.asprj file). If you wish to run Auto-Macros when the XMLSpy project is loaded, check the *Run Auto-Macros* check box.
4. Click **OK** to finish.

Note: To deactivate (that is, unassign) the scripting project of an XMLSpy project, uncheck the *Activate Project Scripts* check box.

21.3.16 Properties



The **Project | Project Properties** command lets you define important settings for any of the specific folders in your project.

To define the Project Properties for a folder:

1. Right-click on the folder you want to define the properties for.
2. Select the **Properties...** command from the context menu.

Please note:

If your project file is under source control, a prompt appears asking if you want to check out the project file (*.spp). Click **OK** if you want to edit settings and be able to save them.

Properties

Name: XML Files

File extensions: xml;cml;math;mtx;rdf;smil;svg:wml

Validation

☐ Validate with: Browse... Window...

XSL transformation of XML files

☒ Use this XSL: C:\Program Files\Altova\xmlspy\Examples\OrgChart.x Browse... Window...

XSL:FO transformation of XML files

☒ Use this XSL: rogram Files\Altova\xmlspy\Examples\OrgChartFO.xsl Browse... Window...

XSL transformation of XSL files

☐ Use this XML: Browse... Window...

Destination files of XSL transformation

☒ Save in folder: C:\Program Files\Altova\xmlspy\Examples Browse...

☐ File extension: .html

Authentic view

☐ Use config.: Browse... Window...

The files specified in the **Use this xxx** entry will take precedence over any local assignment directly within the XML file. For example, the `OrgChart.xsl` file (in the **Use this XSL** entry), will always be used when transforming any of the XML files in the **XML Files** folder. Also, such specified files for individual folders take precedence over files specified for ancestor folders.

File extensions

The File extensions help to determine the automatic file-to-folder distribution that occurs when you add new files to the project (as opposed as to one particular folder).

User ID and password for external folders

Among the properties of external folders (including external Web folders) you can save the User ID and password that might be required for accessing the server.

Validate

Define the DTD or Schema document that should be used to [validate](#) all files in the current folder (Main Pages in this example).

XSL transformation of XML files

You can define the XSL Stylesheet to be used for [XSL Transformation](#) of all files in the folder.

If you are developing XSL Stylesheets yourself, you can also assign an example XML document to be used to preview the XSL Stylesheet in response to an XSL Transformation command issued from the stylesheet document, instead of the XML instance document.

XSL:FO transformation of XML files

You can define the XSL Stylesheet, containing XSL:FO markup, to be used for [XSL:FO](#)

[Transformation](#) of all files in the folder.

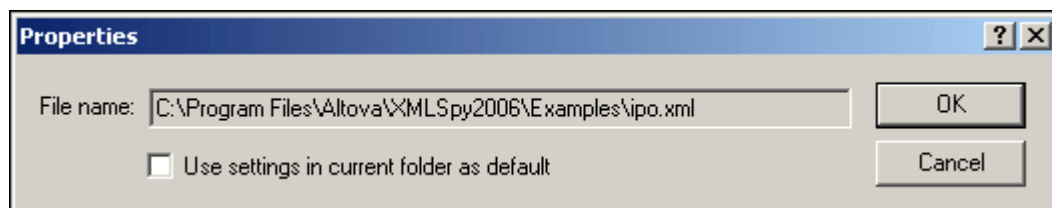
Destination files of XSL transformation

For batch XSL Transformations, you can define the destination directory the transformed files should be placed in.

If you have added one file or URL to more than one folder in your project, you can use the Properties dialog to set the default folder whose settings should be used when you choose to validate or transform the file in non-batch mode. To do this, use the **Use settings in current folder as default** check box (see *screenshot*).

To access the Properties dialog and check this check box:

1. Copy an XML file in a project to a different folder.
2. Right-click the copied file in the Project window and select **Properties** from the context menu.



Authentic View

The "Use config." option allows you to select a StyleVision Power Stylesheet (SPS file) when editing XML files using Authentic View, in the current folder. After you have associated the schema, SPS, and XML files with each other, and entered them in a project, changing the location of any of the files could cause errors among the associations.

To avoid such errors, it is best to finalize the locations of your schema, SPS, and XML files before associating them with each other and assigning them to a project.

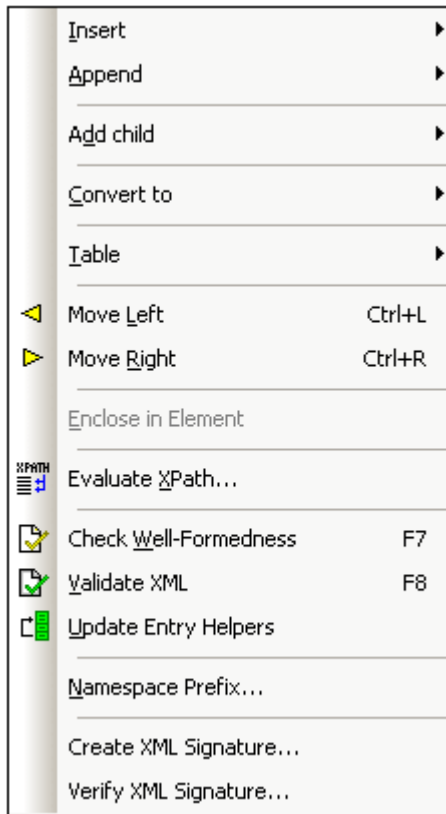
21.3.17 Most Recently Used Projects

This command displays the file name and path for the nine most recently used projects, allowing quick access to these files.

Also note, that XMLSpy can automatically open the [last project](#) that you used, whenever you start XMLSpy. (**Tools | Options | File** tab, Project | Open last project on program start).

21.4 XML Menu

The **XML** menu contains commands commonly used when working with XML documents. You will find commands to insert or append elements, modify the element hierarchy, set a namespace prefix, as well as to evaluate XPath's in the context of individual XML documents.



Among the most frequently used XML tasks are checks for the [well-formedness](#) of documents and [validity](#) of XML documents. Commands for these tasks are in this menu.

21.4.1 Insert

The **XML | Insert** command, though enabled in all views, can be used in Grid View only. It has a submenu (see *screenshot*) with which you can insert:

- The XML declaration and node types (Attribute, Element, Text, CDATA, Comment, Processing Instruction) in XML documents;
- DOCTYPE declarations and external DTD declarations in XML documents;
- DTD declarations (ELEMENT, ATTLIST, ENTITY, and NOTATION) in DTD documents and internal DTD declarations of XML documents.

	A <u>tt</u> ribute	Ctrl+Shift+I
	E <u>l</u> ement	Ctrl+Shift+E
	T <u>e</u> xt	Ctrl+Shift+T
	C <u>D</u> ATA	Ctrl+Shift+D
	C <u>o</u> mment	Ctrl+Shift+M
	X <u>M</u> L	
	P <u>r</u> ocessing Instruction	
	X <u>I</u> nclude...	
	D <u>O</u> CTYPE	
	E <u>x</u> ternalID	
	E <u>L</u> EMENT	
	A <u>T</u> TLIST	
	E <u>N</u> TITY	
	N <u>O</u> TATION	
	Encoded External F <u>i</u> le...	

Insert Attribute



Ctrl+Shift+I

The **XML | Insert | Attribute** command is available in Grid View only, and inserts a new attribute before the selected item. An inserted attribute may appear a few lines before the current item in Grid View. This is because attributes immediately follow their parent element in Grid View and precede all child elements of that parent element.

Insert Element



Ctrl+Shift+E

The **XML | Insert | Element** command is available in Grid View only, and inserts a new element before the selected item. If the current selection is an attribute, the new element is before the first child element of the attribute's parent element.

Insert Text



Ctrl+Shift+T

The **XML | Insert | Text** command is available in Grid View only, and inserts a new text row before the selected item. If the current selection is an attribute, the text row is inserted after the attribute and before the first child element of the attribute's parent element.

Insert CDATA



Ctrl+Shift+D

The **XML | Insert | Cdata** command is available in Grid View only, and inserts a new CDATA block before the selected item. If the current selection is an attribute, the CDATA block is inserted after the attribute and before the first child element of the attribute's parent element.

Insert Comment



Ctrl+Shift+M

The **XML | Insert | Comment** command is available in Grid View only, and inserts a new comment before the selected item. If the current selection is an attribute, the new comment row is inserted after the attribute and before the first child element of the attribute's parent element.

Insert XML



The **XML | Insert | XML** command is available in Grid View only, and inserts a row for the XML declaration before the selected item. You must insert the child attributes of the XML declaration and the values of this attribute. An XML declaration must look something like this:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Please note: Since an XML document may only contain one XML declaration at the very top of the file, this command should only be used with the topmost row selected and if an XML declaration does not already exist.

Insert Processing Instruction

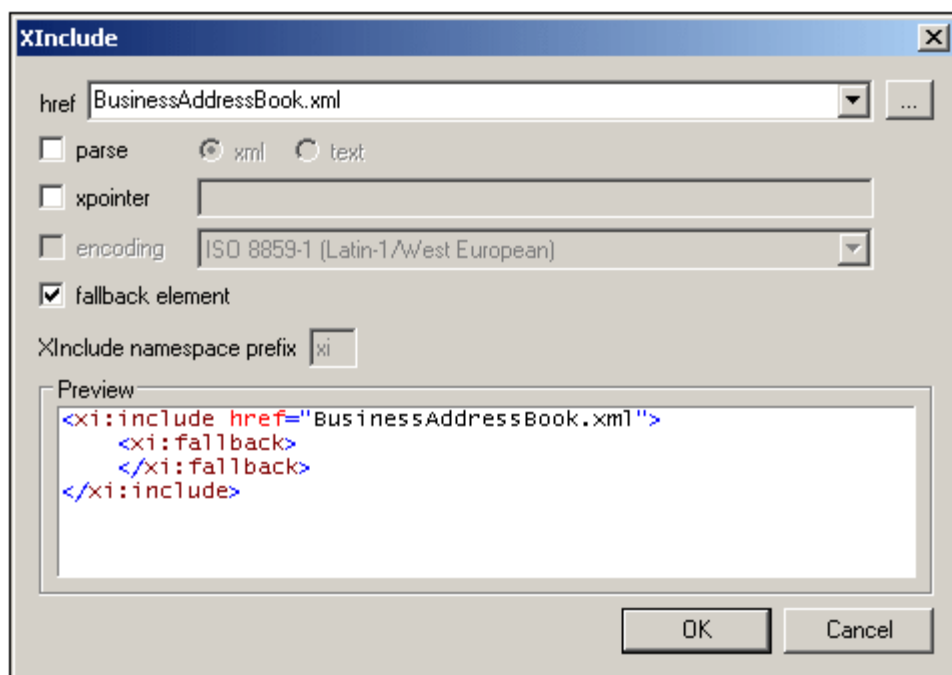


The **XML | Insert | Processing Instruction** command is available in Grid View only, and inserts a new processing instruction (PI) before the selected item. If the current selection is an attribute, the PI is inserted after the attribute and before the first child element of the attribute's parent element.

Insert XInclude



The **XML | Insert | XInclude** command is available in Grid View only, and enables you to insert a new XInclude element before the selected item. If the current selection is an attribute, the XInclude element is inserted after the attribute and before the first child element of the attribute's parent element. Selecting this command pops up the XInclude dialog (*screenshot below*).



The XML file to be included is entered in the `href` text box (alternatively, you can browse for the file by clicking the **Browse (...)** button to the right of the text box). The filename will be entered in the XML document as the value of the `href` attribute. The `parse`, `xpointer`, and `encoding` attributes of the XInclude element (`xi:include`), and the `fallback` child element of `xi:include` can also be inserted via the dialog. Do this by first checking the appropriate check box and then selecting/entering the required values. In the case of the `fallback` element, checking its check box only inserts the empty element. The content of the `fallback` element must be added subsequently in one of the editing views.

The `parse` attribute determines whether the included document is to be parsed as XML or text. (XML is the default value and therefore need not be specified.) The `xpointer` attribute identifies a specific fragment of the document located with the `href` attribute; it is this fragment that will be included. The `encoding` attribute specifies the encoding of the included document so that XMLSpy can transcode this document (or the part of it to be included) into the encoding of the including document. The contents of the `fallback` child element replace the `xi:include` element if the document to be included cannot be located.

Here is an example of an XML document that uses XInclude to include two XML documents:

```
<?xml version="1.0" encoding="UTF-16"?>
<AddressBook xsi:schemaLocation="http://www.altova.com/sv/myaddresses
AddressBook.xsd"
  xmlns="http://www.altova.com/stylevision/tutorials/myaddresses"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xi="http://www.w3.org/2001/XInclude">
  <xi:include href="BusinessAddressBook.xml"/>
  <xi:include href="PersonalAddressBook.xml"/>
</AddressBook>
```

When this XML document is parsed, it will replace the two XInclude elements with the files specified in the respective `href` attributes.

xml: base

When the XML validator of XMLSpy reads an XML document and encounters the `include` element in the XInclude namespace (hereafter `xi:include`), it replaces this element (`xi:include`) with the XML document named in the `href` attribute of the `xi:include` element. The document element (root element) of the included XML document (or the element identified by an XPointer) will be included with an attribute of `xml:base` in order to preserve the base URIs of the included element. If the resulting XML document (containing the included XML document/s or tree fragment/s) must be valid according to a schema, then the document element of the included document (or the top-level element of the tree fragment) must be created with a content model that allows an attribute of `xml:base`. If, according to the schema, the `xml:base` attribute is not allowed on this element, then the resulting document will be invalid. How to define an `xml:base` attribute in an element's content model using XMLSpy's Schema View is described in the [xml: Prefixed Attributes](#) section of the Schema View section of the documentation.

XPointers

XMLSpy supports XPointers in XInclude. The relevant W3C recommendations are the [XPointer Framework](#) and [XPointer element\(\) Scheme](#) recommendations. The use of an XPointer in an XInclude element enables a specific part of the XML document to be included, instead of the entire XML document. XPointers are used within an XInclude element as follows:

```
<xi:include href="PersonalAddressBook.xml" xpointer="element( usa)"/>
<xi:include href="BusinessAddressBook.xml" xpointer="element( /1/1)"/>
<xi:include href="BobsAddressBook.xml" xpointer="element( usa/3/1)"/>
<xi:include href="PatsAddressBook.xml" xpointer="
element( usa) element( /1/1)"/>
```

In the `element()` scheme of XPointer, an NCName or a child sequence directed by integers may be used.

- In the first `xi:include` element listed above, the `xpointer` attribute uses the element scheme with an NCName of `usa`. According to the XPointer Framework, this NCName identifies the element that has an ID of `usa`.
- In the second `xi:include` listed above, the `xpointer` attribute with a value of `element( /1/1)` identifies, in the first step, the first child element of the document root (which, if the document is well-formed, will be its document (or root) element). In the second step, the first child element of the element located in the previous step is located; in our example, this would be the first child element of the document element.
- The `xpointer` attribute of the third `xi:include` listed above uses a combination of NCName and child sequence. This XPointer locates the first child element of the third child element of the element having an ID of `usa`.
- If you are not sure whether your first XPointer will work, you can back it up with a second one as shown in the fourth `xi:include` listed above:
`xpointer="element( usa) element( /1/1) "`. Here, if there is no element with an ID of `usa`, the back-up XPointer specifies that the first child element of the document element is to be selected. Additional backups are also allowed. Individual XPointers may not be separated, or they may be separated by whitespace: for example,
`xpointer="element( usa) element( addresses/1) element( /1/1) "`.

Note: The namespace binding context is not used in the `element()` scheme because the `element()` scheme does not support qualified names.

Insert DOCTYPE



The **XML | Insert | DOCTYPE** command is available in the Grid View of an XML file when a top-level node is selected. It appends a DOCTYPE declaration at the top of the XML document. You must enter the name of the DOCTYPE, and this name must be the same as the name of the document element.



After you have entered the name of the DOCTYPE, you can enter the declarations you wish to use in the internal DTD subset.

Please note:

- A DOCTYPE declaration may only appear between the XML declaration and the XML document element.
- You could use the [Assign DTD](#) command instead to create a DOCTYPE statement that refers to an external DTD document.

Insert ExternalID

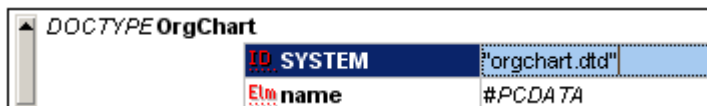


A DOCTYPE declaration in an XML file can contain a reference to an external resource containing DTD declarations. This resource is referenced either through a public or system identifier. For example:

```
<! DOCTYPE doc_element_name PUBLIC "publicID" "systemID">
<! DOCTYPE doc_element_name SYSTEM "systemID">
```

A system identifier is a URI that identifies the external resource. A public identifier is location-independent and can be used to dereference the location of an external resource. For example, in your XMLSpy installation, URIs for popular DTDs and XML Schemas are listed in the catalog files named `catalog.xml` in the various schema folders in `C:\Program Files\Altova\Common2012\Schemas\`. A public identifier in an XML document can be used to dereference a DTD listed in these catalog files.

The **XML | Insert | ExternalID** command is available when a "child" item of the DOCTYPE declaration in an XML file is selected in Grid View. This command inserts a Grid View row for an external identifier (PUBLIC or SYSTEM). You must enter the type of identifier and its value.



The Text View corresponding to the screenshot of the Grid View shown above looks something like this:

```
<! DOCTYPE OrgChart SYSTEM "orgchart.dtd" [
  <! ELEMENT name ( #PCDATA) >
]>
```

Please note: A row for External-ID can be added as a child when the DOCTYPE item is selected, or it can be inserted or appended when one of the child items of the DOCTYPE item is selected, for example, the ELEMENT declaration `name` in the example above.

Insert ELEMENT



The **XML | Insert | ELEMENT** command is available in Grid View only, for DTD documents or when an item in the DOCTYPE declaration of an XML document is selected. It inserts an ELEMENT declaration before the selected declaration.

Insert ATTLIST



The **XML | Insert | ATTLIST** command is available in Grid View only, for DTD documents or when an item in the DOCTYPE declaration of an XML document is selected. It inserts an ATTLIST declaration before the selected declaration.

Insert ENTITY



The **XML | Insert | ENTITY** command is available in Grid View only, for DTD documents or when an item in the DOCTYPE declaration of an XML document is selected. It inserts an ENTITY declaration before the selected declaration.

Insert NOTATION

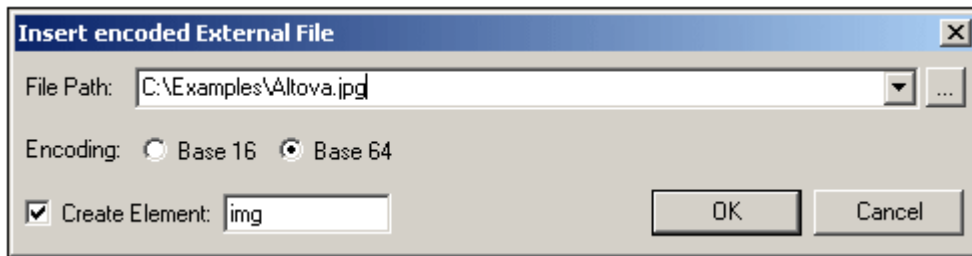


The **XML | Insert | NOTATION** command is available in Grid View only, for DTD documents or when an item in the DOCTYPE declaration of an XML document is selected. It inserts a NOTATION declaration before the selected declaration.

Insert Encoded External File

The **XML | Insert | Encoded External File** command is available in Grid View only. It inserts a binary encoded file, such as an image file, as encoded characters. The encoded external file is inserted before the Grid View selection.

On clicking the command, the Insert Encoded External File dialog (*screenshot below*) pops up. In it you enter the path to the file, select the encoding you want, and specify whether the encoded file is to be inserted in an element or not.



You can browse for or enter the name of the external file to be encoded and embedded. Either a Base-16 or Base-64 encoding must be specified. If you wish to enclose the encoded text in an element, then check the Create Element check box and specify the name of the desired element in the Create Element text box. If the Create Element check box is not checked, then the encoded text will be inserted directly at the cursor location.

On clicking **OK**, the encoded text of the selected file is inserted at the cursor location, with an enclosing element if this has been specified.

The encoded file is inserted in Grid View (*the highlighted element in the screenshot below*).

Person			
	First	Fred	
	Last	Landis	
	Title	Project Manager	
	Phone	123-456-7890	
	Email	f.landis@nanonull.com	
img			
	path	C:\Examples\Altova.jpg	
	encoding	xs:base64Binary	
	Text	/9j/4AAQSkZJRgABAgEASABIAAD/4QqTRXhpZg...	
expense-item (4)			
	type	expto	Date
1	Lodging	Sales	2003-01-01
2	Lodging	Development	2003-01-02
3	Lodging	Marketing	2003-01-02
4	Entertainment	Development	2003-01-02

In Text View, the file will be inserted as below.

```
<img ext="jpg" encoding="xs:base64Binary">
iVBORw0KGgoAAAANSUhEUgAAABAAAAAQMAAAAPW0iAAAAB1BMVEUAAAD/
//+12Z/dAAAM01EQVR4nGP4/5/h/1+G/58ZDrAZ3D/MCh8yw83NDDenge4U
g9C9zwz3gVLMdA/A6P9/AFGGfyjOXZtQAAAAAE1FTkSuQmCC
</img>
```

The listing above shows the encoded text of a JPG image file. An `img` element was created around the encoded text.

21.4.2 Append

The **XML | Append** command, though enabled in all views, can be used in Grid View only. It opens a submenu (see *screenshot*) with which you can append:

- The XML declaration and node types (Attribute, Element, Text, CDATA, Comment,

- Processing Instruction) in XML documents;
- DOCTYPE declarations and external DTD declarations in XML documents
- DTD declarations (ELEMENT, ATTLIST, ENTITY, and NOTATION) in DTD documents and internal DTD declarations of XML documents.

	A <u>tt</u> ribute	Ctrl+Shift+I
	E <u>l</u> ement	Ctrl+Shift+E
	T <u>e</u> xt	Ctrl+Shift+T
	C <u>D</u> ATA	Ctrl+Shift+D
	C <u>o</u> ment	Ctrl+Shift+M
	X <u>M</u> L	
	P <u>r</u> ocessing Instruction	
	X <u>I</u> nclude...	
	D <u>O</u> CTYPE	
	E <u>x</u> ternalID	
	E <u>L</u> EMENT	
	A <u>T</u> TLIST	
	E <u>N</u> TITY	
	N <u>O</u> TATION	
	E <u>n</u> coded External F <u>i</u> le...	

Append Attribute



Ctrl+I

The **XML | Append | Attribute** command is available in Grid View only, and appends a new attribute.

Append Element



Ctrl+E

The **XML | Append | Element** command is available in Grid View only, and appends an element node after the last sibling element of the selected element. If an attribute node is selected, then the element node is appended after the last child of the selected attribute's parent element.

Append Text



Ctrl+T

The **XML | Append | Text** command is available in Grid View only, and appends a text block after the last sibling element of the selected element. If an attribute node is selected, then the text block is appended after the last child of the selected attribute's parent element.

Append CDATA



Ctrl+D

The **XML | Append | Cdata** command is available in Grid View only, and appends a CDATA node after the last sibling of any selected node other than an attribute node. If an attribute node is selected, then the CDATA section is appended after the last child of the selected attribute's parent element.

Append Comment



Ctrl+M

The **XML | Append | Comment** command is available in Grid View only, and appends a comment node after the last sibling of any selected node other than an attribute node. If an attribute node is selected, then the comment node is appended after the last child of the selected attribute's parent element.

Append XML



The **XML | Append | XML** command inserts a new XML declaration `<?xml version="1.0" encoding="UTF-8"?>` as the first item in a document.

Please note: An XML document may contain only one XML declaration, which must appear at the very top of the file.

Append Processing Instruction



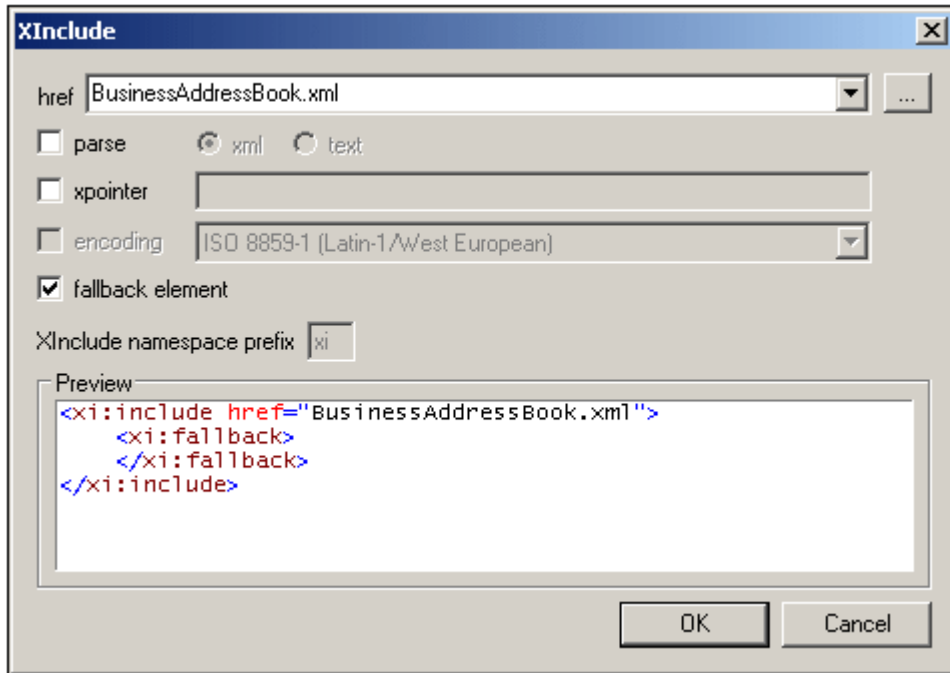
The **XML | Append | Processing Instruction** command is available in Grid View only, and appends a processing instruction node after the last sibling of any selected node other than an attribute node. If an attribute node is selected, then the processing instruction node is appended after the last child of the selected attribute's parent element.

Append XInclude



The **XML | Append | XInclude** command is available in Grid View only, and enables you to append an XInclude element after the last sibling of any selected node other than an attribute

node. If the current selection is an attribute, the XInclude element is appended after the the last child of the selected attribute's parent element. Selecting this command pops up the XInclude dialog (screenshot below).



The XML file to be included is entered in the `href` text box (alternatively, you can browse for the file by clicking the **Browse (...)** button to the right of the text box). The filename will be entered in the XML document as the value of the `href` attribute. The `parse`, `xpointer`, and `encoding` attributes of the XInclude element (`xi:include`), and the `fallback` child element of `xi:include` can also be inserted via the dialog. Do this by first checking the appropriate check box and then selecting/entering the required values. In the case of the `fallback` element, checking its check box only inserts the empty element. The content of the `fallback` element must be added subsequently in one of the editing views.

The `parse` attribute determines whether the included document is to be parsed as XML or text. (XML is the default value and therefore need not be specified.) The `xpointer` attribute identifies a specific fragment of the document located with the `href` attribute; it is this fragment that will be included. The `encoding` attribute specifies the encoding of the included document so that XMLSpy can transcode this document (or the part of it to be included) into the encoding of the including document. The contents of the `fallback` child element replace the `xi:include` element if the document to be included cannot be located.

Here is an example of an XML document that uses XInclude to include two XML documents:

```
<?xml version="1.0" encoding="UTF-16"?>
<AddressBook xsi:schemaLocation="http://www.altova.com/sv/myaddresses
AddressBook.xsd"
  xmlns="http://www.altova.com/stylevision/tutorials/myaddresses"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xi="http://www.w3.org/2001/XInclude">
  <xi:include href="BusinessAddressBook.xml"/>
  <xi:include href="PersonalAddressBook.xml"/>
</AddressBook>
```

When this XML document is parsed, it will replace the two XInclude elements with the files specified in the respective `href` attributes.

xml: base

When the XML validator of XMLSpy reads an XML document and encounters the `include` element in the `XInclude` namespace (hereafter `xi:include`), it replaces this element (`xi:include`) with the XML document named in the `href` attribute of the `xi:include` element. The document element (root element) of the included XML document (or the element identified by an `XPointer`) will be included with an attribute of `xml:base` in order to preserve the base URIs of the included element. If the resulting XML document (containing the included XML document/s or tree fragment/s) must be valid according to a schema, then the document element of the included document (or the top-level element of the tree fragment) must be created with a content model that allows an attribute of `xml:base`. If, according to the schema, the `xml:base` attribute is not allowed on this element, then the resulting document will be invalid. How to define an `xml:base` attribute in an element's content model using XMLSpy's Schema View is described in the [xml: Prefixed Attributes](#) section of the Schema View section of the documentation.

XPointers

XMLSpy supports XPointers in `XInclude`. The relevant W3C recommendations are the [XPointer Framework](#) and [XPointer element\(\) Scheme](#) recommendations. The use of an `XPointer` in an `XInclude` element enables a specific part of the XML document to be included, instead of the entire XML document. XPointers are used within an `XInclude` element as follows:

```
<xi:include href="PersonalAddressBook.xml" xpointer="element( usa)"/>
<xi:include href="BusinessAddressBook.xml" xpointer="element( /1/1)"/>
<xi:include href="BobsAddressBook.xml" xpointer="element( usa/3/1)"/>
<xi:include href="PatsAddressBook.xml" xpointer="
element( usa) element( /1/1)"/>
```

In the `element()` scheme of `XPointer`, an `NCName` or a child sequence directed by integers may be used.

- In the first `xi:include` element listed above, the `xpointer` attribute uses the `element` scheme with an `NCName` of `usa`. According to the `XPointer Framework`, this `NCName` identifies the element that has an ID of `usa`.
- In the second `xi:include` listed above, the `xpointer` attribute with a value of `element( /1/1)` identifies, in the first step, the first child element of the document root (which, if the document is well-formed, will be its document (or root) element). In the second step, the first child element of the element located in the previous step is located; in our example, this would be the first child element of the document element.
- The `xpointer` attribute of the third `xi:include` listed above uses a combination of `NCName` and child sequence. This `XPointer` locates the first child element of the third child element of the element having an ID of `usa`.
- If you are not sure whether your first `XPointer` will work, you can back it up with a second one as shown in the fourth `xi:include` listed above:

```
xpointer="element( usa) element( /1/1) ". Here, if there is no element with an ID of
usa, the back-up XPointer specifies that the first child element of the document element
is to be selected. Additional backups are also allowed. Individual XPointers may not be
separated, or they may be separated by whitespace: for example,
xpointer="element( usa) element( addresses/1) element( /1/1) ".
```

Note: The namespace binding context is not used in the `element()` scheme because the `element()` scheme does not support qualified names.

Append DOCTYPE



The **XML | Append | DOCTYPE** command is available in the Grid View of an XML file when a top-level node is selected. It appends a DOCTYPE declaration at the top of the XML document. You must enter the name of the DOCTYPE, and this name must be the same as the name of the document element.



After you have entered the name of the DOCTYPE, you can enter the declarations you wish to use in the internal DTD subset.

Please note:

- A DOCTYPE declaration may only appear between the XML declaration and the XML document element.
- You could use the [Assign DTD](#) command instead to create a DOCTYPE statement that refers to an external DTD document.

Append ExternalID



A DOCTYPE declaration in an XML file can contain a reference to an external resource containing DTD declarations. This resource is referenced either through a public or system identifier. For example:

```
<!DOCTYPE doc_element_name PUBLIC "publicID" "systemID">
<!DOCTYPE doc_element_name SYSTEM "systemID">
```

A system identifier is a URI that identifies the external resource. A public identifier is location-independent and can be used to dereference the location of an external resource. For example, in your XMLSpy installation, URIs for popular DTDs and XML Schemas are listed in the catalog files named `catalog.xml` in the various schema folders in `C:\Program Files\Altova\Common2012\Schemas\`. A public identifier in an XML document can be used to dereference a DTD listed in these catalog files.

The **XML | Append | ExternalID** command is available when a "child" item of the DOCTYPE declaration in an XML file is selected in Grid View. This command inserts a Grid View row for an external identifier (`PUBLIC` or `SYSTEM`). You must enter the type of identifier and its value.



The Text View corresponding to the screenshot of the Grid View shown above looks something like this:

```
<!DOCTYPE OrgChart SYSTEM "orgchart.dtd" [
  <!ELEMENT name ( #PCDATA) >
]>
```

Please note: A row for External-ID can be added as a child when the DOCTYPE item is selected, or it can be inserted or appended when one of the child items of the DOCTYPE item is selected, for example, the ELEMENT declaration `name` in the example above.

Append ELEMENT



The **XML | Append | ELEMENT** command is available in Grid View only, for DTD documents or when an item in the DOCTYPE declaration of an XML document is selected. It appends an ELEMENT declaration to the list of declarations.

Append ATTLIST



The **XML | Append | ATTLIST** command is available in Grid View only, for DTD documents or when an item in the DOCTYPE declaration of an XML document is selected. It appends an ATTLIST declaration to the list of declarations.

Append ENTITY



The **XML | Append | ENTITY** command is available in Grid View only, for DTD documents or when an item in the DOCTYPE declaration of an XML document is selected. It appends an ENTITY declaration to the list of declarations.

Append NOTATION

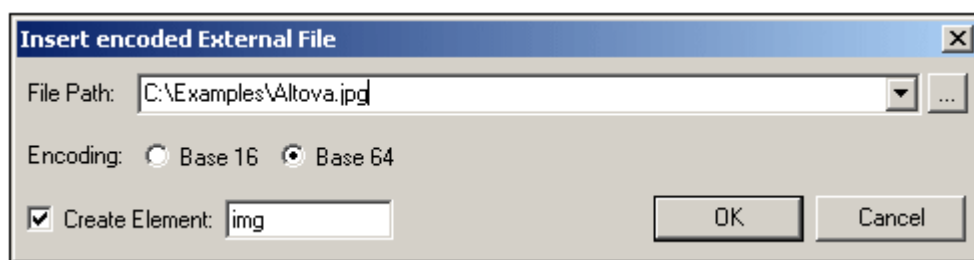


The **XML | Append | NOTATION** command is available in Grid View only, for DTD documents or when an item in the DOCTYPE declaration of an XML document is selected. It appends a NOTATION declaration to the list of declarations.

Append Encoded External File

The **XML | Append | Encoded External File** command is available in Grid View only. It appends a binary encoded file, such as an image file, as encoded characters. The encoded external file is appended after the Grid View selection.

On clicking the command, the Insert Encoded External File dialog (*screenshot below*) pops up. In it you enter the path to the file, select the encoding you want, and specify whether the encoded file is to be inserted in an element or not.



You can browse for or enter the name of the external file to be encoded and embedded. Either a Base-16 or Base-64 encoding must be specified. If you wish to enclose the encoded text in an element, then check the Create Element check box and specify the name of the desired element in the Create Element text box. If the Create Element check box is not checked, then the encoded text will be inserted directly at the cursor location.

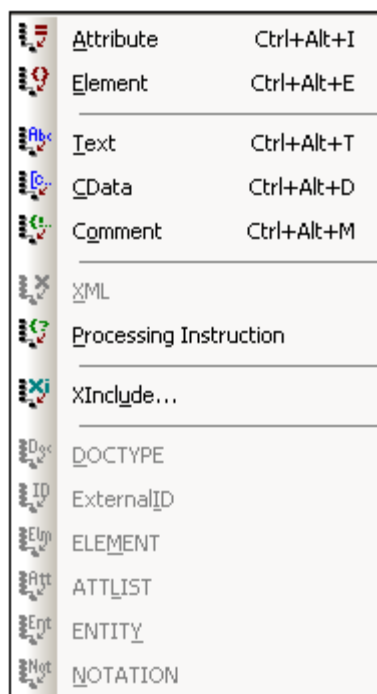
On clicking **OK**, the encoded text of the selected file is inserted at the cursor location, with an enclosing element if this has been specified.

The encoded file is appended in Grid View.

21.4.3 Add Child

The **XML | Add Child** command, though enabled in all views, can be used in Grid View only. It opens a submenu (see screenshot) with which you can add the following child items to the currently selected element.

- The XML declaration and node types (Attribute, Element, Text, CDATA, Comment, Processing Instruction) in XML documents;
- DOCTYPE declarations and external DTD declarations in XML documents
- DTD declarations (ELEMENT, ATTLIST, ENTITY, and NOTATION) in DTD documents and internal DTD declarations of XML documents.



Add Child Attribute



Ctrl+Alt+I

The **XML | Add Child | Attribute** command is available in Grid View only and when an element node is selected. It inserts a new attribute as a child of the selected element node.

Add Child Element



Ctrl+Alt+E

The **XML | Add Child | Element** command is available in Grid View only. It inserts a new element as a child of the selected node.

Add Child Text



Ctrl+Alt+T

The **XML | Add Child | Text** command is available in Grid View only, and inserts new text content as a child of the selected item.

Add Child CDATA



Ctrl+Alt+D

The **XML | Add Child | Cdata** command is available in Grid View only, and inserts a new CDATA section as a child of the selected item.

Add Child Comment



Ctrl+Alt+M

The **XML | Add Child | Comment** command is available in Grid View only, and inserts a new Comment node as a child of the selected item.

Add Child XML



The **XML | Add Child | XML** command is available in Grid View only and when the file is **empty**. It inserts a new XML declaration `<?xml version="1.0" encoding="UTF-8"?>` as the first item in a document.

Please note: An XML document may contain only one XML declaration, which must appear at

the very top of the file.

Add Child Processing Instruction

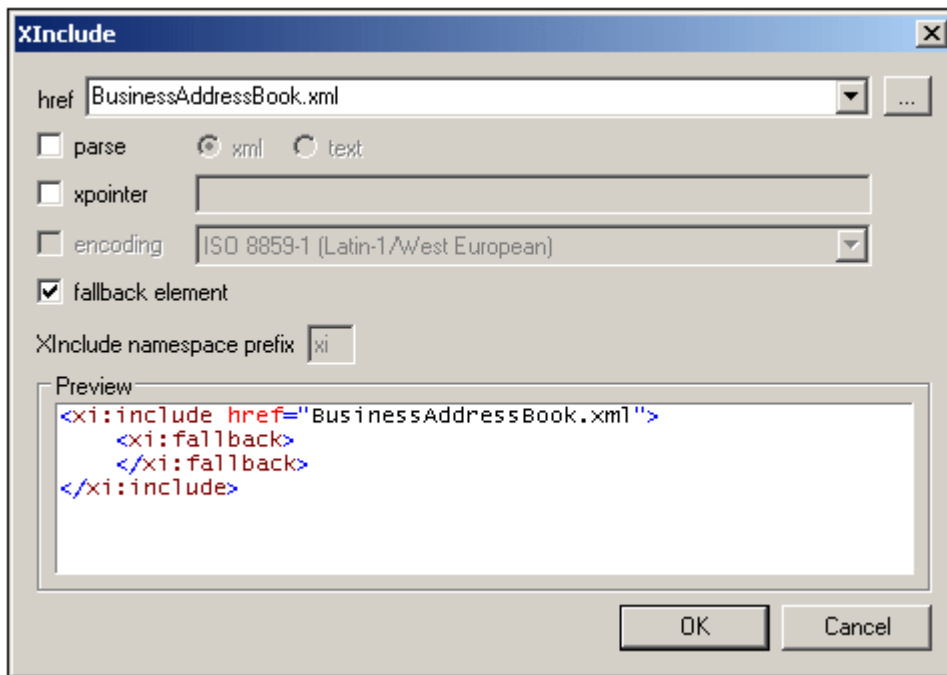


The **XML | Add Child | Processing Instruction** command is available in Grid View only and inserts a new Processing Instruction (PI) as a child of the selected item.

Add Child XInclude



The **XML | Add Child | XInclude** command is available in Grid View only, and enables you to insert an XInclude element as a child of the selected item. Selecting this command pops up the XInclude dialog (*screenshot below*).



The XML file to be included is entered in the `href` text box (alternatively, you can browse for the file by clicking the **Browse (...)** button to the right of the text box). The filename will be entered in the XML document as the value of the `href` attribute. The `parse`, `xpointer`, and `encoding` attributes of the XInclude element (`xi:include`), and the `fallback` child element of `xi:include` can also be inserted via the dialog. Do this by first checking the appropriate check box and then selecting/entering the required values. In the case of the `fallback` element, checking its check box only inserts the empty element. The content of the `fallback` element must be added subsequently in one of the editing views.

The `parse` attribute determines whether the included document is to be parsed as XML or text. (XML is the default value and therefore need not be specified.) The `xpointer` attribute identifies a specific fragment of the document located with the `href` attribute; it is this fragment that will be included. The `encoding` attribute specifies the encoding of the included document so that

XMLSpy can transcode this document (or the part of it to be included) into the encoding of the including document. The contents of the `fallback` child element replace the `xi:include` element if the document to be included cannot be located.

Here is an example of an XML document that uses XInclude to include two XML documents:

```
<?xml version="1.0" encoding="UTF-16"?>
<AddressBook xsi:schemaLocation="http://www.altova.com/sv/myaddresses
AddressBook.xsd"
    xmlns="http://www.altova.com/stylevision/tutorials/myaddresses"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xi="http://www.w3.org/2001/XInclude">
    <xi:include href="BusinessAddressBook.xml"/>
    <xi:include href="PersonalAddressBook.xml"/>
</AddressBook>
```

When this XML document is parsed, it will replace the two XInclude elements with the files specified in the respective `href` attributes.

xml: base

When the XML validator of XMLSpy reads an XML document and encounters the `include` element in the XInclude namespace (hereafter `xi:include`), it replaces this element (`xi:include`) with the XML document named in the `href` attribute of the `xi:include` element. The document element (root element) of the included XML document (or the element identified by an XPointer) will be included with an attribute of `xml:base` in order to preserve the base URIs of the included element. If the resulting XML document (containing the included XML document/s or tree fragment/s) must be valid according to a schema, then the document element of the included document (or the top-level element of the tree fragment) must be created with a content model that allows an attribute of `xml:base`. If, according to the schema, the `xml:base` attribute is not allowed on this element, then the resulting document will be invalid. How to define an `xml:base` attribute in an element's content model using XMLSpy's Schema View is described in the [xml: Prefixed Attributes](#) section of the Schema View section of the documentation.

XPointers

XMLSpy supports XPointers in XInclude. The relevant W3C recommendations are the [XPointer Framework](#) and [XPointer element\(\) Scheme](#) recommendations. The use of an XPointer in an XInclude element enables a specific part of the XML document to be included, instead of the entire XML document. XPointers are used within an XInclude element as follows:

```
<xi:include href="PersonalAddressBook.xml" xpointer="element( usa)"/>
<xi:include href="BusinessAddressBook.xml" xpointer="element( /1/1)"/>
<xi:include href="BobsAddressBook.xml" xpointer="element( usa/3/1)"/>
<xi:include href="PatsAddressBook.xml" xpointer="
element( usa) element( /1/1)"/>
```

In the `element()` scheme of XPointer, an NCName or a child sequence directed by integers may be used.

- In the first `xi:include` element listed above, the `xpointer` attribute uses the element scheme with an NCName of `usa`. According to the XPointer Framework, this NCName identifies the element that has an ID of `usa`.
- In the second `xi:include` listed above, the `xpointer` attribute with a value of `element( /1/1)` identifies, in the first step, the first child element of the document root (which, if the document is well-formed, will be its document (or root) element). In the second step, the first child element of the element located in the previous step is located; in our example, this would be the first child element of the document element.

- The `xpointer` attribute of the third `xi:include` listed above uses a combination of NCName and child sequence. This XPointer locates the first child element of the third child element of the element having an ID of `usa`.
- If you are not sure whether your first XPointer will work, you can back it up with a second one as shown in the fourth `xi:include` listed above:
`xpointer="element(usa) element(/1/1) "`. Here, if there is no element with an ID of `usa`, the back-up XPointer specifies that the first child element of the document element is to be selected. Additional backups are also allowed. Individual XPointers may not be separated, or they may be separated by whitespace: for example,
`xpointer="element(usa) element(addresses/1) element(/1/1) "`.

Note: The namespace binding context is not used in the `element()` scheme because the `element()` scheme does not support qualified names.

Add Child DOCTYPE



The **XML | Add Child | DOCTYPE** command is only available in the Grid View of an **empty** document. It inserts a DOCTYPE declaration in an XML document. The DOCTYPE declaration can be used to declare an internal DTD subset.

Add Child ExternalID



The **XML | Add Child | ExternalID** command is available only when the DOCTYPE declaration of an XML file is selected in Grid View. This command inserts a Grid View row for an external identifier (**PUBLIC** or **SYSTEM**). You must enter the type of identifier and its value.



The Text View corresponding to the screenshot of the Grid View shown above looks something like this:

```
<! DOCTYPE OrgChart SYSTEM "orgchart.dtd" [
  <! ELEMENT name ( #PCDATA) >
]>
```

Please note: A row for External-ID can be added as a child when the DOCTYPE item is selected, or it can be inserted or appended when one of the child items of the DOCTYPE item is selected, for example, the `ELEMENT` declaration `name` in the example above.

Add Child ELEMENT



The **XML | Add Child | ELEMENT** command is available in Grid View only, for DTD documents, or when the DOCTYPE declaration of an XML document is selected. It appends an `ELEMENT` declaration to the list of declarations.

Add Child ATTLIST



The **XML | Add Child | ATTLIST** command is available in Grid View only, for DTD documents, or when the DOCTYPE declaration of an XML document is selected. It appends an ATTLIST declaration to the list of declarations.

Add Child ENTITY



The **XML | Add Child | ENTITY** command is available in Grid View only, for DTD documents, or when the DOCTYPE declaration of an XML document is selected. It appends an ENTITY declaration to the list of declarations.

Add Child NOTATION

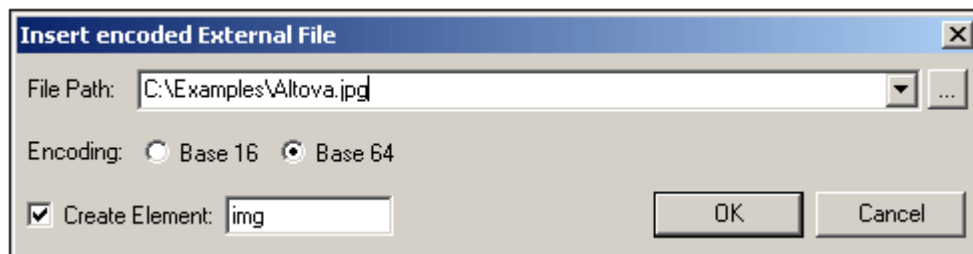


The **XML | Add Child | NOTATION** command is available in Grid View only, for DTD documents, or when the DOCTYPE declaration of an XML document is selected. It appends an NOTATION declaration to the list of declarations.

Add Child Encoded External File

The **XML | Add Child | Encoded External File** command is available in Grid View only. It adds a binary encoded file as a child node. The encoded external file is inserted as a child of the Grid View selection.

On clicking the command, the Insert Encoded External File dialog (*screenshot below*) pops up. In it you enter the path to the file, select the encoding you want, and specify whether the encoded file is to be inserted in an element or not.

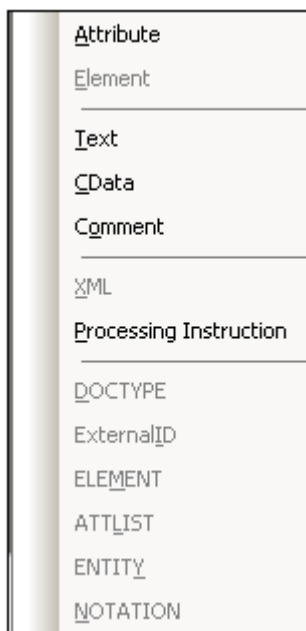


On clicking **OK**, the encoded text of the selected file is inserted at the cursor location, with an enclosing element if this has been specified.

The encoded file is added as a child in Grid View.

21.4.4 Convert To

The **XML | Convert to** command converts a selected item in Grid View to a different item type. This operation is available only in Grid View on individual items that do not contain any child node. Placing the cursor over the **Convert to** command displays a submenu (see screenshot) which contains the items to which the selected item can be converted.



Please note: If the operation you select would result in a loss of data (for example, converting an attribute to a comment would result in a loss of the attribute name), a warning dialog box will appear.

Convert To Attribute

The **XML | Convert to | Attribute** command converts the selected item to a attribute.

Convert To Element

The **XML | Convert to | Element** command converts the selected item to an element.

Convert To Text

The **XML | Convert to | Text** command converts the selected item into text content.

Convert To CDATA

The **XML | Convert to | Cdata** command converts the selected item into a CDATA segment.

Convert To Comment

The **XML | Convert to | Comment** command converts the selected item into a comment.

Convert To XML

The **XML | Convert to | XML** command converts the selected item to an XML declaration:

```
<?xml version="1.0" encoding="UTF-8"?>.
```

Please note: Each XML document may only contain one XML declaration and it must appear at the very top of the file.

Convert To Processing Instruction

The **XML | Convert to | Processing Instruction** command converts the selected item to a new Processing Instruction (PI).

Convert To DOCTYPE

The **XML | Convert to | DOCTYPE** command converts the selected item to a DOCTYPE declaration (in an XML file).

Please note: A DOCTYPE declaration may only appear at the top of an XML instance document between the XML Declaration and the document element of the XML document.

Convert To ExternalID

The **XML | Convert to | ExternalID** command converts the selected item to an external DTD reference in a DOCTYPE declaration (`PUBLIC` or `SYSTEM` identifier).

Convert To ELEMENT

The **XML | Convert to | ELEMENT** command converts the selected item to an element declaration in a DOCTYPE declaration or in an external DTD.

Convert To ATTLIST

The **XML | Convert to | ATTLIST** command converts the selected item to an attribute list declaration in a DOCTYPE declaration or in an external DTD.

Convert To ENTITY

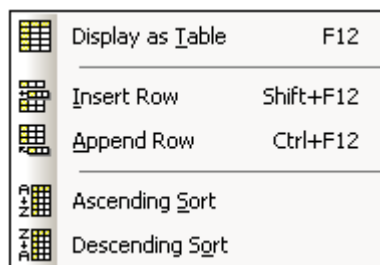
The **XML | Convert to | ENTITY** command converts the selected item to an entity declaration in a DOCTYPE declaration or in an external DTD.

Convert To NOTATION

The **XML | Convert to | NOTATION** command converts the selected item to a notation declaration in a DOCTYPE declaration or in an external DTD.

21.4.5 Table

The **XML | Table** command, though enabled in all views, can be used only in Grid View. It displays a submenu with all the commands relevant to the [Database/Table View](#) of Grid View.



Display as Table

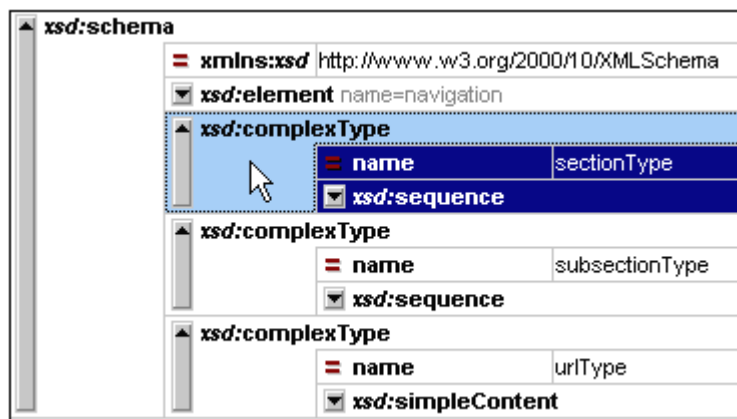


F12

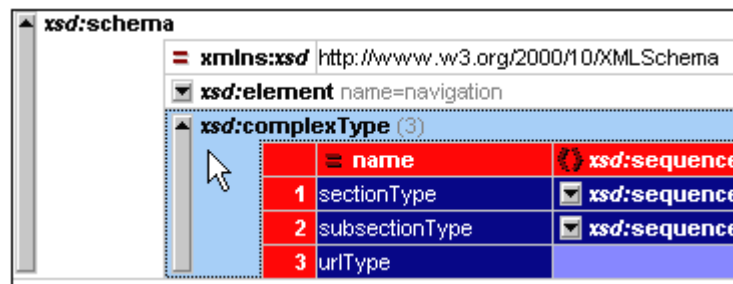
The **XML | Table | Display as Table** command allows you to switch between the standard [Grid View](#) and [Database/Table View](#) (or Table View) of a document element. The Table View enables you to view repeated elements as a table in which the rows represent the occurrences while the columns represent child nodes (including comments, CDATA sections, and PIs).

To switch to Table View:

1. Select any one occurrence of the repeating element you wish to view as a table.



2. Click **XML | Table | Display as Table** or F12 or the toolbar icon.



The element is displayed as a table and the toolbar icon is activated.

To switch from the Table View of a document element to the normal Grid View of that element, select the table or any of its rows or columns, and click the toolbar icon. That table element switches to Grid View.

Note: Table View colors can be set in the Colors tab of the Options dialog (**Tools | Options | Colors**)

Insert Row



Shift+F12

The **XML | Table | Insert Row** command is enabled in [Database/Table View](#) when a row or cell is selected. It inserts a new row before the selected row. The new row corresponds to an occurrence of the table element. Mandatory child elements are created for the new element

Append Row



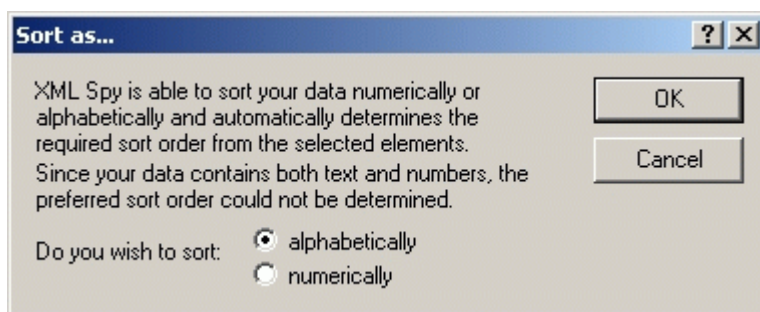
Ctrl+F12

The **XML | Table | Append Row** command is enabled in [Database/Table View](#) when a row or cell is selected. It appends a new row after the last row of the table. The new row corresponds to an occurrence of the table element. Mandatory child elements are created for the new element

Ascending Sort



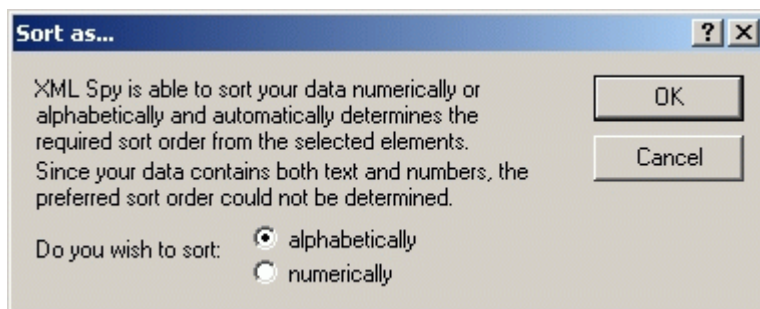
The **XML | Table | Ascending Sort** command is enabled in [Database/Table View](#) when a column or cell is selected. It sorts the column in either alphabetic or numeric ascending order. XMLSpy tries to automatically determine what kind of data is used in the column, and sorts on alphabetic or numeric order, as required. In case of uncertainty, you will be prompted for the sort method to use (see [screenshot](#)).



Descending Sort



The **XML | Table | Descending Sort** command is enabled in [Database/Table View](#) when a column or cell is selected. It sorts the column in either alphabetic or numeric descending order. XMLSpy tries to automatically determine what kind of data is used in the column, and sorts on alphabetic or numeric order, as required. In case of uncertainty, you will be prompted for the sort method to use (see *screenshot*).



21.4.6 Move Left



Ctrl+L

The **XML | Move Left** command is available in Grid View only. It moves the selected node to the left by one level, thereby changing a child element into a sibling of its parent. This command is often referred to as the **Promote** command.

21.4.7 Move Right



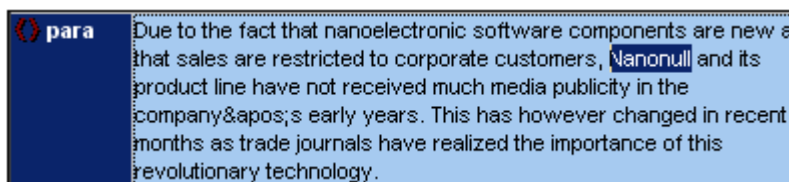
Ctrl+R

The **XML | Move Right** command is available in Grid View only. It moves the selected node to the right by one level, thereby turning it into a child element of the preceding sibling element. This command is often referred to as the **Demote** command.

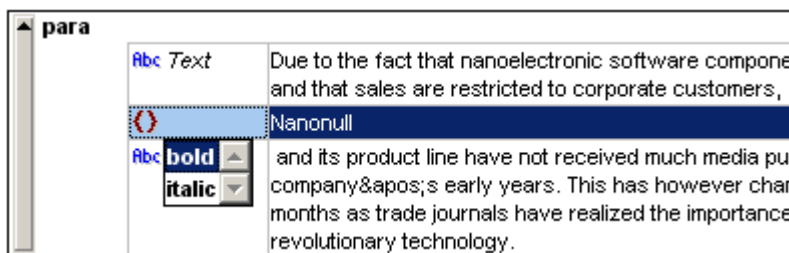
21.4.8 Enclose in Element

The **XML | Enclose in Element** command is enabled in Grid View only. It encloses a selected text range in a new element. The new element is created inline around the selected text. If you are editing a document based on a Schema or DTD, you will automatically be presented with a list of valid choices for the name of the element in which the text is to be enclosed.

For example, in the screenshot below, the text `Nanonull` in the `para` element is highlighted.



When you select the command **XML | Enclose in Element**, the text `Nanonull` is enclosed in a newly created inline element and a list appears offering a choice of `bold` or `italic` for the name of the element. These elements are defined in the schema as children of `para`.



The selection you make will be the name of the new element. Alternatively, you can enter some other name for the element.

21.4.9 Evaluate XPath



The **XML | Evaluate XPath** command opens the Output Windows if these are not open and activates the [XPath tab in the Output Windows](#). In the XPath tab, you can evaluate an XPath expression on the active document and see the results in the Output Window.

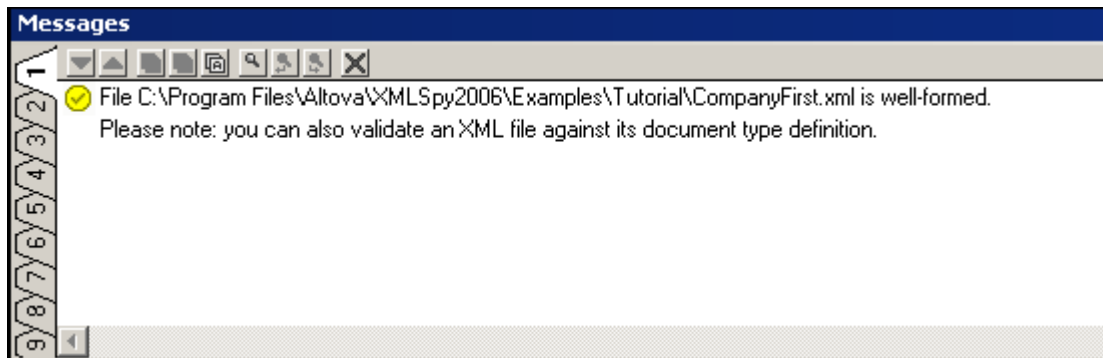
21.4.10 Check Well-Formedness



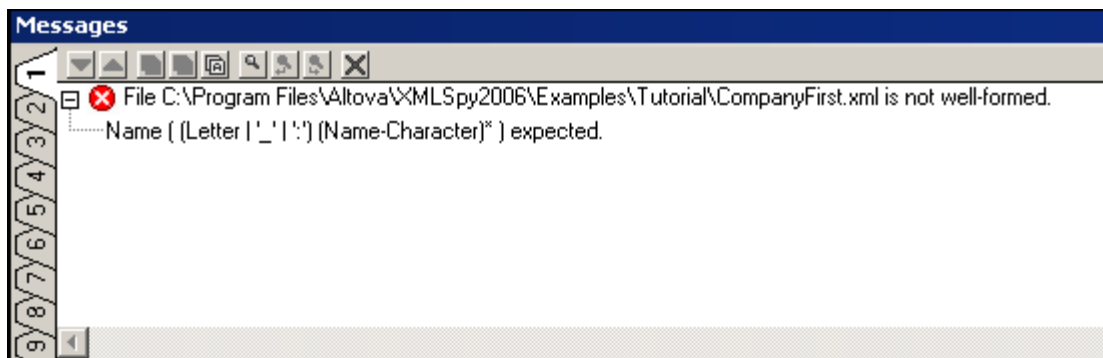
F7

The **XML | Check well-formedness (F7)** command checks the active document for well-formedness by the definitions of the XML 1.0 specification. Every XML document **must** be well-formed. XMLSpy checks for well-formedness whenever a document is opened or saved, or when the view is changed from Text to any other view. You can also check for well-formedness at any time while editing by using this command.

If the well-formedness check succeeds when you explicitly invoke the check, a message is displayed in the Validation window:



If an error is encountered during the well-formedness check, a corresponding error message is displayed:



Please note: The output of the Messages window has 9 tabs. The validation output is always displayed in the active tab. Therefore, you can check well-formedness in tab1 for one schema file and keep the result by switching to tab 2 before validating the next schema document (otherwise tab 1 is overwritten with the validation result).

It is generally not permitted to save a malformed XML document, but XMLSpy gives you a Save Anyway option. This is useful when you want to suspend your work temporarily (in a not well-formed condition) and resume it later.

Please note: You can also use the **Check well-formedness** command on any file, folder, or group of files in the active [project window](#). Click on the respective item, and then on the Check Well-Formedness icon.

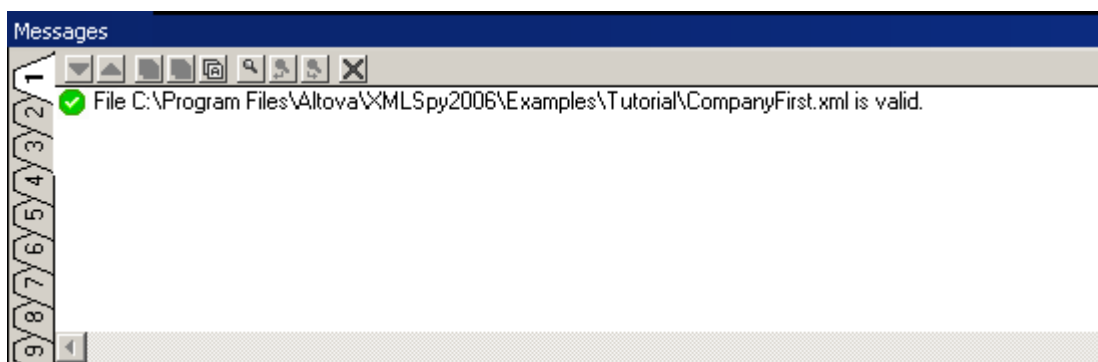
21.4.11 Validate XML



F8

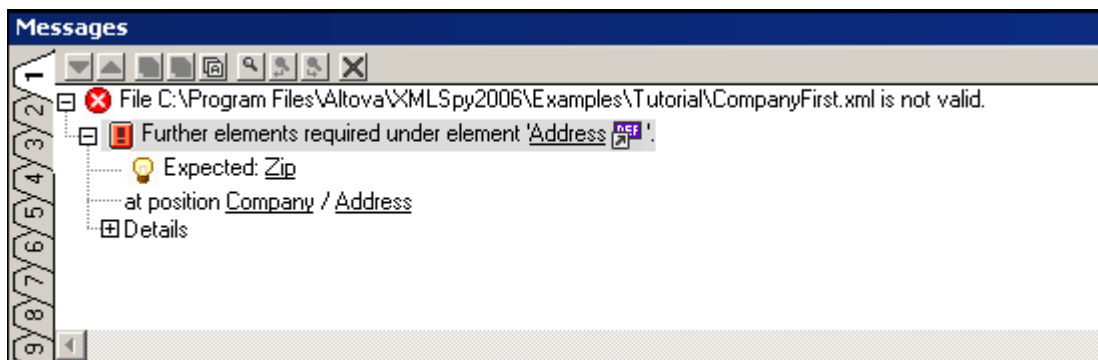
The **XML | Validate (F8)** command enables you to validate XML documents against DTDs, XML Schemas, and other schemas. Validation is automatically carried out when you switch from Text View to any other view. You can specify that a document be automatically validated when a file is opened or saved (**Tools | Options | File**). The **Validate** command also carries out a well-formedness check before checking validity, so there is no need to use the Check Well-Formedness command before using the **Validate** command.

If a document is valid, a successful validation message is displayed in the Validation window:



Otherwise, a message that describes the error is displayed.

Please note: You can click on the links in the error message to jump to the spot in the XML file where the error was found.



Please note: The output of the Validation window has 9 tabs. The validation output is always displayed in the active tab. Therefore, you can check well-formedness in tab1 for one schema file and keep the result by switching to tab 2 before validating the next schema document (otherwise tab 1 is overwritten with the validation result).

The command is normally applied to the active document. But you can also apply the command to a file, folder, or group of files in the active project. Select the required file or folder in the Project Window (by clicking on it), and click **XML | Validate** or **F8**. Invalid files in a project will be opened and made active in the Main Window, and the File Is Invalid error message will be displayed.

Validating XML documents

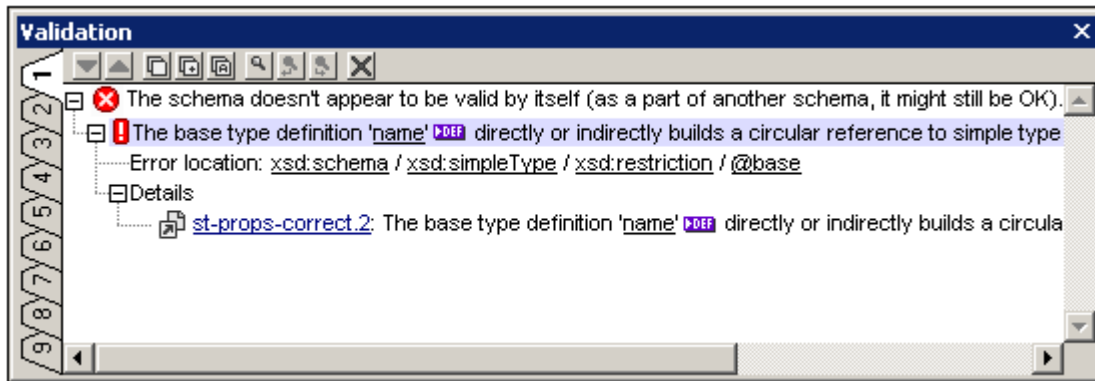
To validate an XML file, make the XML document active in the Main Window, and click **XML | Validate** or **F8**. The XML document is validated against the schema referenced in the XML file. If no reference exists, an error message is displayed in the Messages Window. As long as the XML document is open, the schema is kept in memory (see [Flush Memory Cache](#) in the DTD/Schema menu).

Validating schema documents (DTDs and XML Schema)

XMLSpy supports major schema dialects, including DTD and XML Schema. To validate a schema document, make the document active in the Main Window, and click **XML | Validate** or **F8**.

Schema validation messages in the Validation window

If the schema (DTD or XML Schema) is valid, a successful validation message is displayed in the Validation window; the Validation window pops up at the bottom of the application window. If the schema is not valid, one or more error messages are displayed in the Validation window (*screenshot below*).



The error message is divided into three parts:

1. A description of the error. The description contains links to the relevant declarations or definitions.
2. The location of the error within the schema structure. Clicking a node in the location path highlights that node in the document.
3. Details of the error provides more information about the error and contains a link to the relevant paragraph in the relevant specification (the XML specification for DTD validation; and XML Schema specification for XML Schema validation).

Please note: When the validation is done in Text View, clicking a link in the Validation window highlights the corresponding declaration or definition in Text View. When the validation is done in Schema View, the corresponding declaration or definition is highlighted either in the design window of Schema View or in the relevant entry helper window.

Catalogs

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

- XMLSpy loads a file called `RootCatalog.xml`, which contains a list of catalog files that will be looked up. You can enter as many catalog files to look up, each in a `nextCatalog` element in `RootCatalog.xml`.
- The catalog files included in `RootCatalog.xml` are looked up and the URIs are resolved according to the mappings specified in the catalog files. You should take care not to duplicate mappings, as this could lead to errors.
- Two catalog files are supplied with XMLSpy. How these work is described in the section [Catalogs in XMLSpy](#).
- The `PUBLIC` or `SYSTEM` identifier in the `DOCTYPE` statement of your XML file will be used for the catalog lookup. For popular schemas, the `PUBLIC` identifier is usually pre-defined, thus requiring only the URI in the catalog file to be changed when XML documents are used on multiple machines.

When writing your `CustomCatalog.xml` file (or other custom catalog file), use only the following

subset of the OASIS catalog in order for XMLSpy to process the catalog correctly. Each of the elements in the supported subset can take the `xml:base` attribute, which is used to specify the base URI of that element.

```
<catalog...>
...
<public publicId="PublicID of Resource" uri="URL of local file"/>
<system systemId="SystemID of Resource" uri="URL of local file"/>
<rewriteURI uriStartString="StartString of URI to rewrite"
rewritePrefix="String to replace StartString"/>
<rewriteSystem systemIdStartString="StartString of SystemID"
rewritePrefix="Replacement string to locate resource locally"/>
<uri name="filename" uri="URL of file identified by filename"/>
...
</catalog>
```

Please note:

- The `catalog.xml` file in the `%AltovaCommonFolder%\Schemas\schema` folder contains references to DTDs that implement older XML Schema specifications. You should not validate your XML Schema documents against any of these schemas. The referenced DTD files are included solely to provide XMLSpy with entry helper info for editing purposes should you wish to create documents according to these older recommendations. *Also see next point.*
- If you create a custom file extension for a particular schema (for example, the `.myhtml` extension for (HTML) files that are to be valid according to the HTML DTD), then you can enable intelligent editing for files with these extensions by adding a line of text to `CustomCatalog.xml`. For the example extension mentioned, you should add the element `<spy:fileExtHelper ext="myhtml" uri="schemas/xhtml/xhtml11-transitional.dtd"/>` as a child of the `<catalog>` element. This would enable intelligent editing (auto-completion, entry helpers, etc) of `.myhtml` files in XMLSpy according to the XHTML 1.0 Transitional DTD.
- For more information on catalogs, see the [XML Catalogs specification](#).

Automating validation with Altova XML 2012

AltovaXML is a free application which contains Altova's XML Validator, XSLT 1.0, XSLT 2.0, and XQuery 1.0 engines. It can be used from the command line, via a COM interface, in Java programs, and in .NET applications to validate XML documents, transform XML documents using XSLT 1.0 and 2.0 stylesheets, and execute XQuery documents.

Validation tasks can therefore be automated with the use of Altova XML. For example, you can create a batch file that calls AltovaXML to perform validation on a set of documents and sends the output to a text file. See the [AltovaXML documentation](#) for details.

21.4.12 Validating WSDL Files

The following list contains important information about WSDL validation behavior in the Enterprise and Professional Editions of XMLSpy:

- The Professional Edition performs only simple schema validation, i.e., it treats the WSDL file as an XML file and validates it according to the appropriate schema defined at <http://schemas.xmlsoap.org/wsdl/>.
- The Enterprise Edition provides WSDL validation that goes beyond the validation provided by the Professional Edition. It does not validate against <http://schemas.xmlsoap.org/wsdl/> but rather uses the

http://www.altova.com/specs_wsdl.html#_document-s as well as its own logic to provide more valuable validation information in the context of WSDL. Thus it can happen that a WSDL file is valid in the Professional Edition, but not valid in the Enterprise Edition (see Example).

- There is a difference between <http://schemas.xmlsoap.org/wsdl/> and http://www.altova.com/specs_wsdl.html#_document-s. <http://schemas.xmlsoap.org/wsdl/> is missing the extensibility elements (the specification and the schema should be the same but are not; this difference appears to be an error in the schema).
- The fact that the Professional Edition uses <http://schemas.xmlsoap.org/wsdl/> for validation means that any extensibility elements will be considered invalid. The Enterprise Edition used http://www.altova.com/specs_wsdl.html#_document-s, which allows extensibility elements. Thus a WSDL file containing extensibility elements is valid when using the Enterprise edition, and not valid when using the Professional Edition.
- The fact that the file is invalid in the Professional Edition is because we are using the official schema provided by the W3C working group. Any errors in this schema are beyond our control.

Example

The following example is part of a WSDL file. Notice the element "getCityTime" that has been declared in the file. This element is mistakenly referenced as "getCityTimes". The Enterprise Edition checks if elements that are referenced have previously been declared in the file; the Professional Edition does not. This file (assuming that the rest of the file is valid) would be found to be valid in the Professional Edition, but not valid in the Enterprise Edition (assuming that "getCityTimes" is not defined somewhere else in the file).

```
<s:element name="getCityTime">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="city"
type="s:string"/>
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="abc">
  <s:complexType>
    <s:sequence>
      <s:element ref="getCityTimes"/>
    </s:sequence>
  </s:complexType>
</s:element>
```

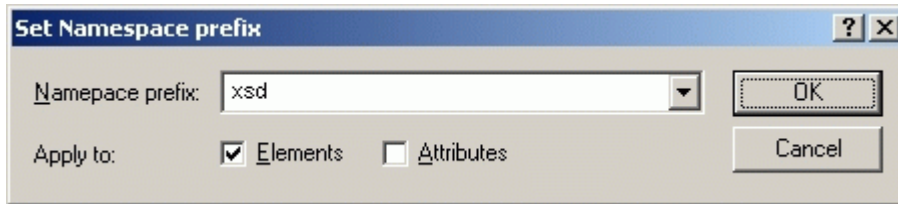
21.4.13 Update Entry Helpers



The **XML | Update Entry Helpers** command updates the Entry Helper windows by reloading the underlying DTD or Schema. If you have modified the XML Schema or DTD that an open XML document is based upon, it is advisable to update the Entry Helpers so that the intelligent editing information reflects the changes in the schema.

21.4.14 Namespace Prefix.

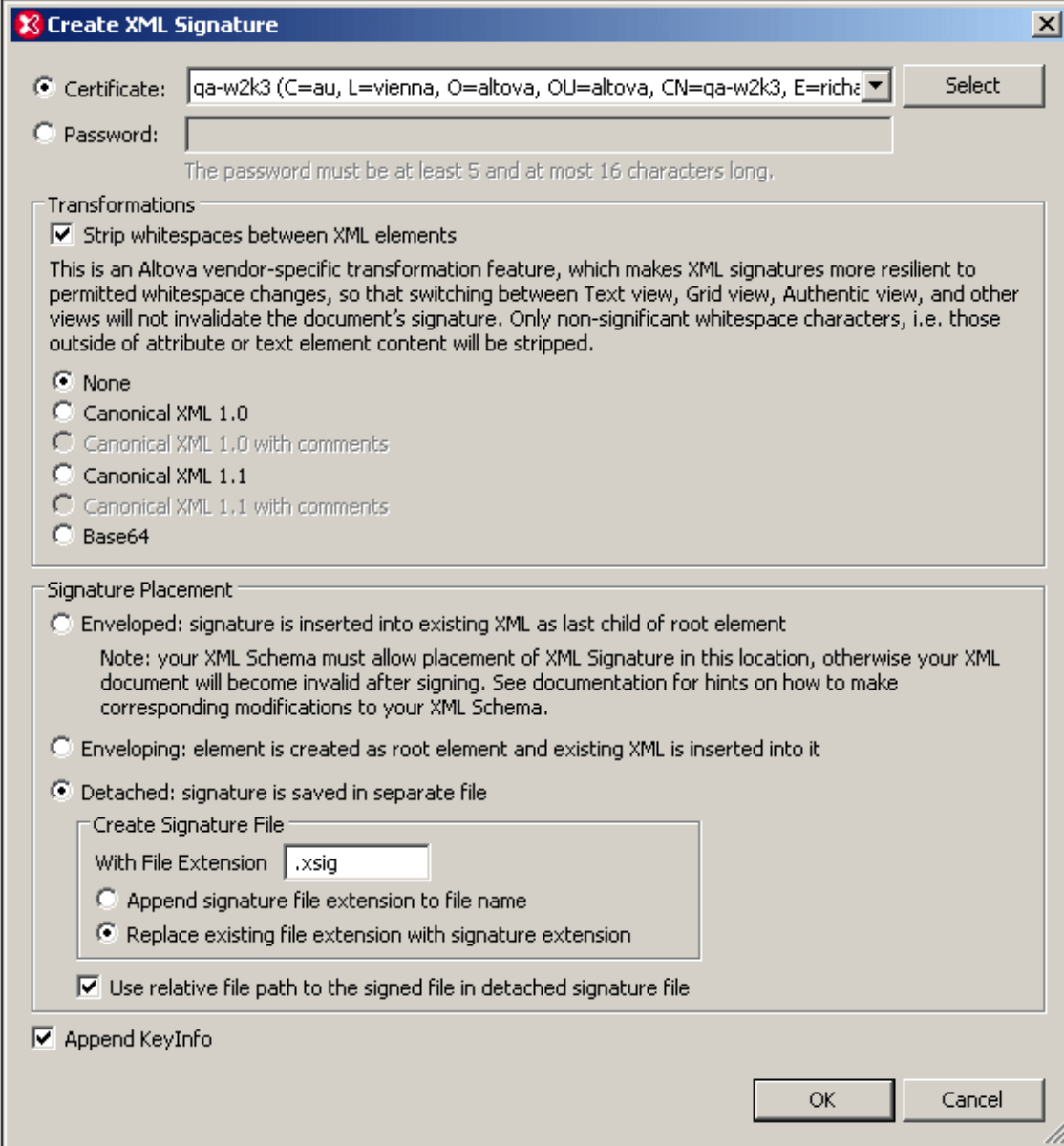
The **XML | Namespace Prefix...** command is available in Grid View and opens a dialog box in which you can set the namespace prefix of the selected element or attribute, and, in the case of elements, of its descendants as well.



You can choose to set the namespace prefix on either elements, attributes, or both. The namespace prefix is applied to the selected element or attribute, and, if an element is selected, to descendant nodes of the selected element.

21.4.15 Create XML Signature

The **Create XML Signature** command is enabled in Text View, Grid View, Schema View, WSDL View and XBRL View, and enables you to create an XML signature for the active XML document. Clicking the command opens the Create XML Signature dialog (*screenshot below*), the settings of which are explained below.



The dialog box is titled "Create XML Signature". It has two radio buttons at the top: "Certificate:" and "Password:". The "Certificate:" option is selected, and its text box contains "qa-w2k3 (C=au, L=vienna, O=altova, OU=altova, CN=qa-w2k3, E=richa...". To the right of this text box is a "Select" button. Below the "Password:" radio button is an empty text box, and below that is a note: "The password must be at least 5 and at most 16 characters long."

The "Transformations" section has a checked checkbox "Strip whitespaces between XML elements". Below this is a descriptive text: "This is an Altova vendor-specific transformation feature, which makes XML signatures more resilient to permitted whitespace changes, so that switching between Text view, Grid view, Authentic view, and other views will not invalidate the document's signature. Only non-significant whitespace characters, i.e. those outside of attribute or text element content will be stripped." Below this text are five radio buttons: "None" (selected), "Canonical XML 1.0", "Canonical XML 1.0 with comments", "Canonical XML 1.1", and "Canonical XML 1.1 with comments". At the bottom of this section is a radio button for "Base64".

The "Signature Placement" section has three radio buttons: "Enveloped: signature is inserted into existing XML as last child of root element", "Enveloping: element is created as root element and existing XML is inserted into it", and "Detached: signature is saved in separate file" (selected). Below the "Detached" option is a sub-section "Create Signature File" containing a text box "With File Extension" with ".xsig" entered. Below this text box are two radio buttons: "Append signature file extension to file name" and "Replace existing file extension with signature extension" (selected). Below this sub-section is a checked checkbox "Use relative file path to the signed file in detached signature file".

At the bottom of the dialog is a checked checkbox "Append KeyInfo". At the bottom right are "OK" and "Cancel" buttons.

Authentication method: certificate or password

The signature can be based on a certificate or a password. Select the radio button of the method you wish to use.

- **Certificate:** If you wish to use a certificate, the certificate must have a private key and be located in an accessible [certificate store](#). The signature is generated using the private key of the certificate. To verify the signature, access to the certificate (or a public-key version of it) is required. The public key of the certificate is used to verify the signature. To select the private-public-key certificate you wish to use, click the **Select** button and browse for the certificate. For more details about certificates, see the section [Working with Certificates](#).
- **Password:** Enter a password with a length of five to 16 characters. This password will subsequently be required to verify the signature.

Transformations

The XML data is transformed and the result of the transformation is used for the creation of the signature. You can specify the canonicalization algorithm to be applied to the file's XML data (the `SignedInfo` content) prior to performing signature calculations. Significant points of difference between the algorithms are noted below:

- *Canonical XML with or without comments*: If comments are included for signature calculation, then any change to comments in the XML data will result in verification failure. Otherwise, comments may be modified or be added to the XML document after the document has been signed, and the signature will still be verified as authentic.
- *Base64*: The root (or document) element of the XML document is considered to be Base64 encoded, and is read in its binary form. If the root element is not Base64, an error is returned or the element is read as empty, depending on what type of element is encountered.
- *None*: No transformation is carried out and the XML data from the binary file saved on disk is passed directly for signature creation. Any subsequent change in the data will result in a failed verification of the signature. However, if the *Strip Whitespace* check box option is selected, then all whitespace is stripped and changes in whitespace will be ignored. A major difference between the *None* option and a *Canonicalization* option is that canonicalization produces an XML data stream, in which some differences, such as attribute order, are normalized. As a result, a canonicalization transformation will normalize any changes such as that of attribute order (so verification will succeed), while no-transformation will reflect such a change (verification will fail).

Signature placement

The signature can be placed within the XML file or be created as a separate file. The following options are available:

- *Enveloped*: The signature element is created as the last child element of the root (document) element.
- *Enveloping*: The signature element is created as the root (document) element and the XML document is inserted as a child element.
- *Detached*: The XML signature is created as a separate file. In this case, you can specify the file extension of the signature file and whether the file name is created with: (i) the extension appended to the name of the XML file (for example, `test.xml.xsig`), or (ii) the extension replacing the XML extension of the XML file (for example, `test.xsig`). You can also specify whether, in the signature file, the reference to the XML file is a relative or an absolute path.

Note: XML signatures for XML Schema (.xsd) files and for XBRL files can only be created as external signature files. For WSDL files, signatures can be created as external files and can be "enveloped" in the WSDL file.

Note: If the XML signature is created as a separate file, then the XML file and signature file are associated with each other via a reference in the signature file. Consequently, signature verification in cases where the signature is in an external file must be done with the signature file active—not with the XML file active.

Append key information

The *Append Keyinfo* option is available when the signature is certificate-based. It is unavailable if the signature is password-based.

If the option is selected, public-key information is placed inside the signature, otherwise key information is not included in the signature. The advantage of including key information is that

the certificate itself (specifically the public-key information in it) will not be required for the verification process (since the key information is present in the signature).

21.4.16 Verify XML Signature

An XML signature will be correctly verified if the XML file has not been changed since having been signed. Otherwise the verification will fail. The **Verify XML Signature** command executes the verification process and displays the results of the verification in the Messages windows. The various verification scenarios in XMLSpy are described below:

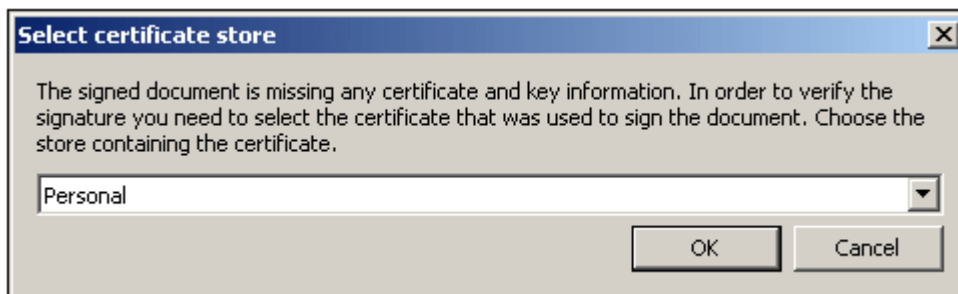
- [XML file contains certificate-based signature, key information included in signature](#)
- [XML file contains certificate-based signature, key information not contained in signature](#)
- [Certificate-based signature in external file, key information contained in signature](#)
- [Certificate-based signature in external file, key information not contained in signature](#)
- [XML file contains password-based signature](#)
- [Password-based signature in external file](#)

XML file contains certificate-based signature, key information included in signature

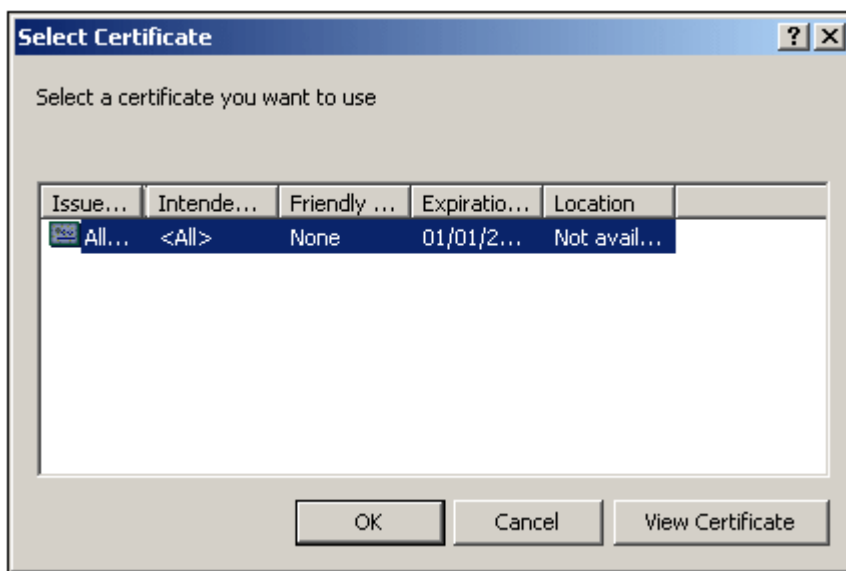
To verify the XML signature in this scenario, make the XML file active in XMLSpy. On clicking the **XML | Verify XML Signature** command, the verification process will be executed and the result will be displayed in the Messages window (verification succeeded or failed).

XML file contains certificate-based signature, key information not contained in signature

If no key information is contained in the certificate-based signature, XMLSpy will prompt you for the certificate from which public-key information for the verification can be read. Verification is done with the XML file active in XMLSpy. On clicking the **XML | Verify XML Signature** command, you will be prompted to select the [certificate store](#) in which the certificate is stored (*screenshot below*).



On selecting a [certificate store](#) and clicking **OK**, a dialog displaying the certificates in that store pops up (*screenshot below*). Select the certificate required for the verification and click **OK**.



The signature is verified and the result is displayed in the Messages window.

Certificate-based signature in external file, key information contained in signature

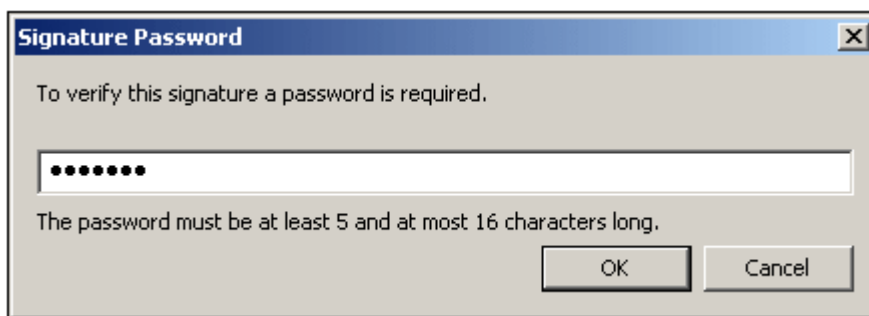
If a certificate-based XML signature is in an external file, the signature is verified with the signature file active in XMLSpy. On clicking the **XML | Verify XML Signature** command, the verification process will be executed and the result will be displayed in the Messages window (verification succeeded or failed).

Certificate-based signature in external file, key information not contained in signature

If a certificate-based XML signature is in an external file, the signature is verified with the signature file active in XMLSpy. On clicking the **XML | Verify XML Signature** command, XMLSpy will prompt you for the certificate from which public-key information for the verification can be read. Select the certificate as described in the section: [XML file contains certificate-based signature, key information not contained in signature](#). The verification process will be executed and the result will be displayed in the Messages window (verification succeeded or failed).

XML file contains password-based signature

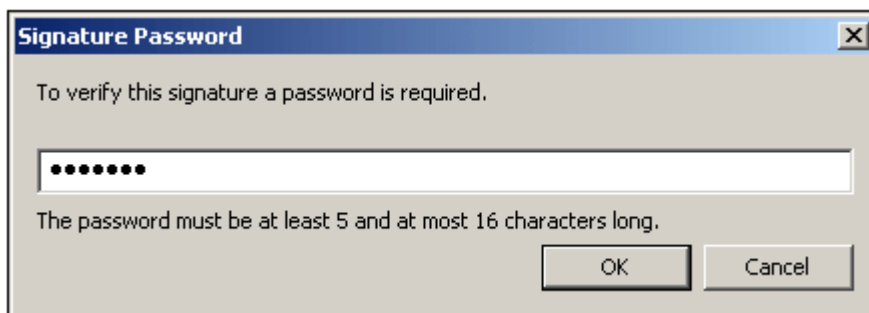
If the XML file contains a password-based XML signature, the signature is verified with the XML file active in XMLSpy. On clicking the **XML | Verify XML Signature** command, a dialog pops up prompting you for the password (*screenshot below*).



Enter the password, which must be five to sixteen characters long, and then click **OK**. The verification process will be executed and the result will be displayed in the Messages window (verification succeeded or failed).

Password-based signature in external file

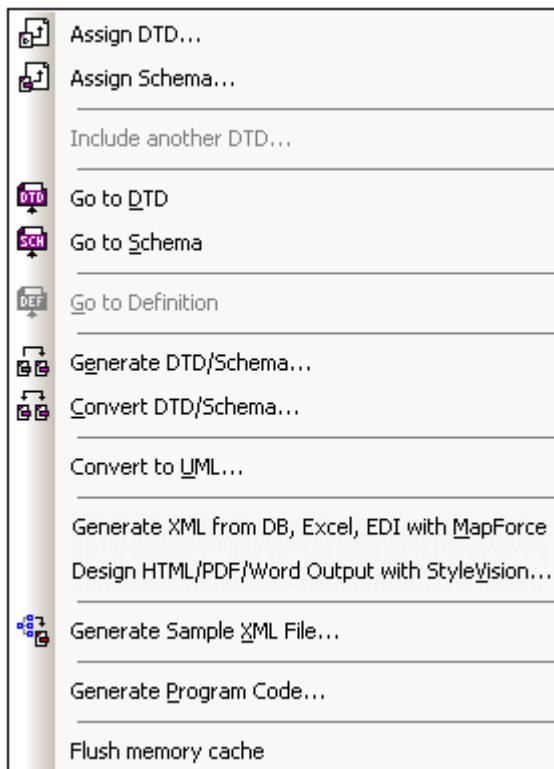
If a password-based XML signature is in an external file, the signature is verified with the signature file active in XMLSpy. On clicking the **XML | Verify XML Signature** command, a dialog pops up prompting you for the password (*screenshot below*).



Enter the password, which must be five to sixteen characters long, and then click **OK**. The verification process will be executed and the result will be displayed in the Messages window (verification succeeded or failed).

21.5 DTD/Schema Menu

The **DTD/Schema** menu contains commands that let you work efficiently with DTDs and XML Schemas.



This section contains a complete description of all the commands in this menu.

21.5.1 Assign DTD



The **DTD/Schema | Assign DTD...** command is enabled when an XML file is active. It assigns a DTD to an XML document, thus allowing the document to be validated and enabling intelligent editing for the document. The command opens the Assign File dialog to let you specify the DTD file you wish to assign. You can also select a file via a global resource or a URL (click the [Browse](#) button) or a file in one of the open windows in XMLSpy (click the **Window** button). Note that you can make the path of the assigned DTD file relative by clicking the Make Path Relative To... check box. When you are done, your XML document will contain a DOCTYPE declaration that references the assigned DTD. The DOCTYPE declaration will look something like this:

```
<!DOCTYPE main SYSTEM "http://link.xmlspy.com/spyweb.dtd">
```

Please note: A DTD can be assigned to a new XML file at the time the file is created.

21.5.2 Assign Schema



The **DTD/Schema | Assign Schema...** command is enabled when an XML document is active. It assigns an XML Schema to an XML document, thus allowing the document to be validated and enabling intelligent editing for the document. The command opens the Assign File dialog to let you specify the XML Schema file you wish to assign. You can also select a file via a global resource or a URL (click the [Browse](#) button) or a file in one of the open windows in XMLSpy (click the **Window** button). Note that you can make the path of the assigned file relative by clicking the Make Path Relative To... check box. When you are done, your XML document will contain an XML Schema assignment with the required namespaces. The schema assignment will look something like this:

```
xmlns="http://www.xmlspy.com/schemas/icon/orgchart"  
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"  
xsi:schemaLocation="http://www.xmlspy.com/schemas/icon/orgchart  
http://schema.xmlspy.com/schemas/icon/orgchart.xsd"
```

21.5.3 Include Another DTD

The **DTD/Schema | Include another DTD...** command allows you to include another Document Type Definition (DTD) or external parsed entity into the internal subset of a document type definition, or in any DTD document. This is done by defining a corresponding external parsed entity declaration and using that entity in the following line:

```
<! ENTITY % navigation.dtd SYSTEM "S:\xml\navigation.dtd">  
%navigation.dtd;
```

The command opens the Assign File dialog to let you specify the DTD file you want to include in your DTD.

Please note: This command is enabled in Grid View only.

21.5.4 Go to DTD



The **DTD/Schema | Go to DTD** command opens the DTD on which the active XML document is based. If no DTD is assigned, then an error message is displayed.

21.5.5 Go to Schema



The **DTD/Schema | Go to Schema** command opens the XML Schema on which the active XML document is based. If no XML Schema is assigned, then an error message is displayed.

21.5.6 Go to Definition



The **DTD/Schema | Go to Definition** command displays the exact definition of an element or attribute in the corresponding Document Type Definition or Schema document.

To see the item definition in Grid View

1. Click left on the item.
2. Select the menu item **DTD/Schema | Go to Definition**, or click on the icon.

To see the item definition in Schema View

- Use CTRL + Double click on the item you want to see the definition of, or
- Click the item and select menu option **DTD/Schema | Go to Definition**, or click on the icon.

In both cases, the corresponding DTD or Schema file is opened, and the item definition is highlighted.

21.5.7 Generate DTD/Schema



The **DTD/Schema | Generate DTD/Schema** command generates a new DTD or W3C XML Schema from an XML document (or from a set of XML documents contained in a folder in the project window). This command is useful when you want to generate a DTD or XML Schema from XML documents.

Generate DTD/Schema ? X

DTD/Schema file format

☐ DTD

☒ W3C Schema

OK

Cancel

☒ Generate one shared type for all equal named elements

☒ Validate and resolve entities

Define types used for elements

☒ Local (if applicable)

☐ Global

Define simple types used for attributes

☐ Global, merge equal types into one

☐ As distinct global types

☒ Local

Define attributes with same name and type

☒ Local

☐ Global

Simple type recognition

☐ Best possible

☐ Numbers only

☒ No detection

Create enumerations for

☒ All types of values

☐ Plain strings only

☒ Always

☐ For a maximum distinct values

Ignore values longer than characters for enumerations

If you generate an XML Schema, the following options are available:

- **Elements:** The type of elements can be defined locally or globally (*Define types for elements*). If elements have the same name, a common type can be declared for use in the definition of these elements (*Generate one shared type*).
- **Attributes:** The simple types of attributes (*Define simple types for attributes*) can be defined as (i) common global types; (ii) distinct global types; (iii) local types. Attributes with the same name and type can be defined either locally or globally.
- **Simple type recognition:** The recognition of types (*Simple type recognition*) can be set to: (i) best possible; (ii) recognition of number datatypes only; (iii) no datatype recognition, in which case all datatypes are set to `xs:string`.
- **Entity resolution:** In the XML document, entities may appear in element content and attribute values. Whether they are resolved or not (*Validate and resolve entities*) is therefore significant for enumeration values. Furthermore, some entities (especially parsed entities that contain markup) can affect the content model differently depending on whether they are resolved or not. Note that the XML document will be validated for being correct XML before the schema is generated. If the document is invalid, the schema generation process will be discontinued.
- **Enumerations:** All types of values, or string values only, can be enumerated.

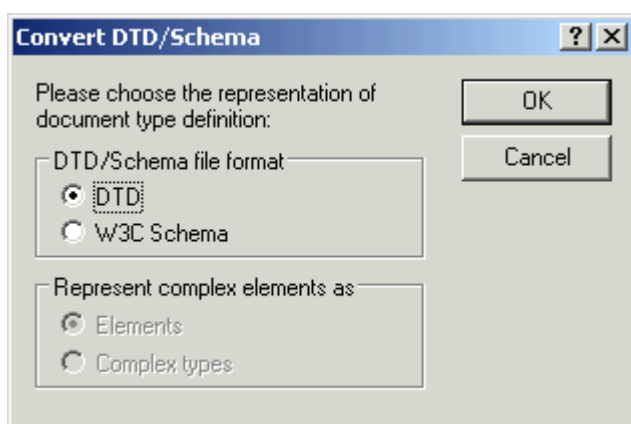
If you generate a DTD, the entity resolution and enumeration options are available.

The Generate DTD/Schema command normally operates on the active main window, but you can also use the **Generate DTD/Schema** command on any file, folder, or group of files in the active project window.

21.5.8 Convert DTD/Schema



The **DTD/Schema | Convert DTD/Schema...** command converts a DTD into an XML Schema and vice versa. If you select W3C Schema, you must also specify how to represent complex elements, as elements or as complex types.



Converting a DTD to XML Schema

When you convert a DTD to XML Schema, XMLSpy makes a few assumptions because of the limited information available. Most notably, the values of certain DTD components are treated literally rather than having their semantics parsed. This is because the program cannot know which of several possible usages is intended. In these cases, you should modify the generated conversion.

In any case, you should carefully examine the generated conversion to see if you can enhance it. A few areas in which improvements may be required are given below.

Attribute Datatyping

DTDs allow for only 10 attribute datatypes, whereas XML Schemas, for instance, allow for 44 datatypes plus derived datatypes. You may wish to enhance a generated XML Schema, for example, by using a more restrictive datatype. Note that when an XML Schema is converted to DTD datatype information will be lost.

Namespaces

DTDs are not namespace-aware. As a result, if namespaces are to be specified in a DTD they must be hard-coded into element and attribute names. This could pose challenging problems when converting from one schema to another.

Entities

XML Schema does not have equivalents for the general entity declarations of DTDs. When XMLSpy converts a DTD to an XML Schema, it ignores entity declarations.

Unparsed data declarations

DTDs and XML Schemas use different mechanisms for handling unparsed data. This is explained in more detail below.

DTDs use the following mechanism:

1. A notation is declared consisting of a name and an identifier, e.g.
`<! NOTATION gif SYSTEM "image/gif">`
2. You declare the entity, e.g.
`<! ENTITY cover_img SYSTEM "graphics/cover_img.gif" NDATA gif>`
3. Typically, you specify an attribute type of ENTITY on the relevant attribute, e.g.
`<! ELEMENT img EMPTY>`
`<! ATTLIST img format ENTITY #REQUIRED>`

In XML Schema, the corresponding mechanism is as follows:

1. Declare a notation. This functions in the same way as for the DTD.
`<xs:notation name="gif" public="image/gif"/>`
 Note that the `public` attribute is mandatory and holds the identifier. An optional `system` attribute holds the system identifier and is usually an executable that can deal with resources of the notation type.
2. You associate the notation declaration with a given attribute value using the `NOTATION` datatype.
 - You cannot, however, use the `NOTATION` datatype directly, but must derive another datatype from the `NOTATION` datatype.
`<xs:simpleType name="formatType">`
`<xs:restriction base="xs:NOTATION">`
`<xs:enumeration value="gif"/>`
`<xs:enumeration value="jpeg"/>`
`</xs:restriction>`
`</xs:simpleType>`
 - You associate the attribute with the datatype derived from the `NOTATION` datatype, e.g.
`<xs:complexType name="imgType">`
`<xs:attribute name="height"/>`
`<xs:attribute name="width"/>`
`<xs:attribute name="location"/>`
`<xs:attribute name="format" type="formatType"`
`use="required"/>`
`</xs:complexType>`
`<xs:element name="img" type="imgType"/>`

When you convert a DTD to an XML Schema using the Convert DTD/Schema command, XMLSpy does the following:

- Something like
`<! ATTLIST image format ENTITY #REQUIRED`
`...>`
 is converted to
`<xs:attribute name="format" type="xs:ENTITY" use="required"/>`
- And
`<! NOTATION gif SYSTEM "image/gif">`
 is converted to
`<xs:notation name="gif" system="image/gif"/>`

You should therefore make the following modifications:

1. In notations like `<xs:notation name="gif" system="image/gif"/>` replace `system` with `public`, and add an optional system identifier if required.
2. Derive a datatype from the `NOTATION` datatype as described above for `formatType`.
3. Associate the derived datatype with the relevant attribute.

Please note: According to the XML Schema specification, you do not need to—or cannot, depending on your viewpoint—declare an external entity.

Converting an XML Schema to a DTD

When you convert an XML Schema to a DTD, the **namespace prefixes** used in the XML Schema—not the namespace URIs or the namespace declarations—are carried through to the names of the corresponding elements and attributes in the DTD.

Note the following points:

1. Since XML parsers ignore namespaces when validating an XML document against a DTD, the namespace declarations themselves are not converted.
2. The `elementFormDefault` and the `attributeFormDefault` attributes of the `xs:schema` element determine what elements and attributes have their prefixes included in the conversion process. If set to `unqualified`, then only globally declared elements and attributes, respectively, include prefixes in the conversion. If set to `qualified`, all element and attribute names have their prefixes included in the conversion.
3. Prefixes are converted to their corresponding string value plus a colon. Elements and attributes in default namespaces are converted to elements and attributes with names that begin with the string: `default_NS_X`, where `X` is an integer (starting with 1 and having a maximum value equal to the number of default namespaces used in the XML Schema).
4. In the DTD, element names are composed of parameter entities. This enables you to easily change the prefix in the DTD should the prefix in the XML document ever need to change. Parameter entity definitions can be changed either in the DTD document itself or by overriding the parameter entity definitions in the XML document's internal DTD subset.

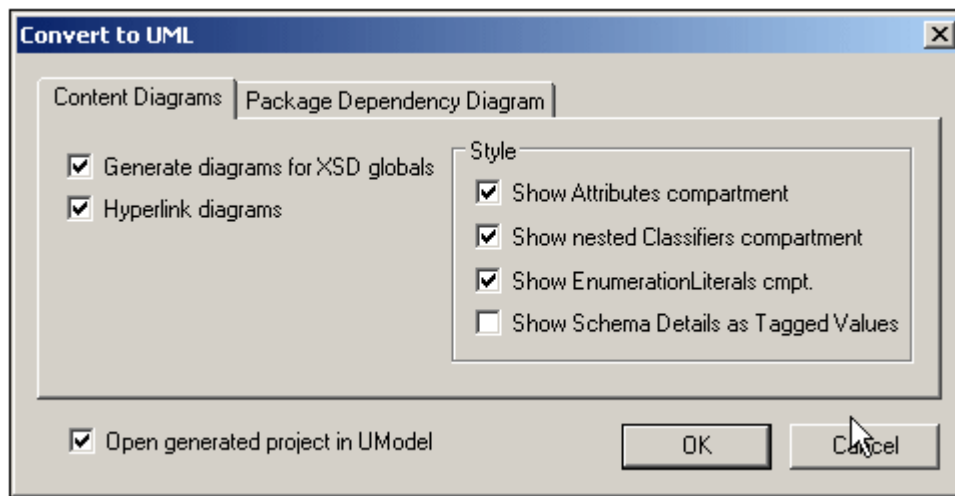
Please note: Namespaces have no semantic value in DTDs, and namespace prefixes carried over from the XML Schema become merely a lexical part of the name of the element or attribute defined in the DTD.

21.5.9 Convert to UML

The **DTD/Schema | Convert to UML** command converts a W3C XML Schema to an Altova UModel Project (`.ump`) document (hereafter UModel project). UMP is the native format of Altova UModel, Altova's UML modeling application. UMP files can then be viewed and edited in Altova UModel.

To convert a schema to UML, do the following:

1. With the schema open, click the **Convert to UML** command. This pops up the Convert to UML dialog (*screenshot below*).



2. In the Content Diagrams tab, select the option Generate Diagrams for XSD Globals. This will generate, in the UModel project, a content model diagram for each global component.
3. Select the required options from those available in the dialog. These options are explained below.
4. If you wish to view the created project in UModel immediately, select the option to open the project in UModel. Otherwise leave this option unselected.
5. Click **OK**.
6. In the Save As dialog that appears, browse for the destination folder, then enter the name of the UMP file, and click **Save**.

Convert to UML options

The following options are available in the Convert to UML dialog.

In the **Content Diagrams** tab:

- *Hyperlink diagrams* creates in each diagram a link to the entry of that global component in the Model Tree view, thus enabling the component to be quickly located in the schema hierarchy.
- In the *Style* pane, the show compartments options enables various compartments to be either shown or hidden.

In the **Package Dependency Diagram** tab:

- The *Generate Diagram* option determines whether a package dependency diagram is generated. A package dependency diagram provides an overview of the entire package, showing the relationships of package components to one another. Note that the other options in this tab will be enabled only if the Generate Diagram option is selected.
- Selecting the *Hyperlink Package to Diagram* option creates a link from the package diagram to the Model Tree View.
- Four options are available for the layout of the package dependencies diagram: (i) unorganized layout (Autolayout option unselected); (ii) hierarchical layout (Autolayout and Hierarchical options selected); (iii) block (Autolayout and Block options selected); and (iv) evenly spaced (Autolayout and Force Directed options selected). The layout can be modified by editing the diagram in UModel.

Note: The Convert to UML feature supports W3C XML Schemas only.

21.5.10 Generate XML from DB, Excel, EDI with MapForce

The **DTD/Schema | Generate XML from DB, Excel, EDI with MapForce** command launches Altova's MapForce if the application is installed. MapForce enables you to map a schema to another DTD, XML Schema, or database and to generate XML.

21.5.11 Design HTML/PDF/Word Output with StyleVision...

The **DTD/Schema | Design HTML/PDF Output in StyleVision...** command launches Altova's StyleVision if the application is installed. StyleVision enables you to design stylesheets for HTML, PDF, and RTF output.

21.5.12 Generate Sample XML File

The **DTD/Schema | Generate Sample XML File** command generates an XML file based on the currently active schema (DTD or XML Schema) in the main window.

Generate sample XML file

☐ Minimize number of elements

☐ Generate mandatory elements only

☒ Generate all elements

OK

Cancel

Generate elements if marked repeatable in Schema/DTD

☐ Generate non-mandatory attributes

☒ Fill elements with data

☐ Fill attributes with data

☐ Treat element content of nillable elements as non-mandatory

☒ For elements with an abstract type, try to use a non-abstract type for xsi:type

Assign schema/DTD to the generated document

☒ With a relative path

☐ With an absolute path

☐ Not at all

☒ Use manually added sample values, if available

☒ Randomize

☐ Cycle through

☐ Use first

Please select root

☒ Elements in namespace http://www.xmlspy.com/schemas/orgchart

Department

Desc

Email

Name

OrgChart

Person

para

Elements in namespace http://www.xmlspy.com/schemas/orgchart

Elements to be generated

One of the following choices can be selected: (i) minimize the number of elements, which generates the minimum number of elements required to create a valid file; (ii) generate mandatory elements only; (iii) generate all elements (whether mandatory or non-mandatory). If elements are defined as being repeatable, the number of elements to be generated (for each of these repeatable elements) can be specified.

Generate non-mandatory attributes

Activating this option generates not only mandatory attributes, but also the non-mandatory attributes, defined in the schema.

Generate X elements if marked repeatable in Schema/DTD

Activating this option generates the number of repeatable elements you enter in the text box.

Fill elements and attributes with data

Activating this option inserts the data type descriptors/values for the respective elements/attributes. For example: `Boolean = 1`, `xsd:string = string`, `Max/Min inclusive = the value defined in the schema`.

Nilable elements and abstract types

The contents of nilable elements can be treated as non-mandatory, and elements with an abstract type can use a non-abstract type for its `xsi:type` attribute.

Schema assignment for the generated XML file

The schema used to generate the XML file can be assigned to the generated XML file with a relative or absolute path.

Use manually added sample values if available

If the schema component has sample values assigned to it, then these will be used as the value or content of that component. For individual components, sample values are assigned in the [Facets Entry Helper](#), in the Samples tab. Which value from the available sample values is selected for a single file generation can be specified:

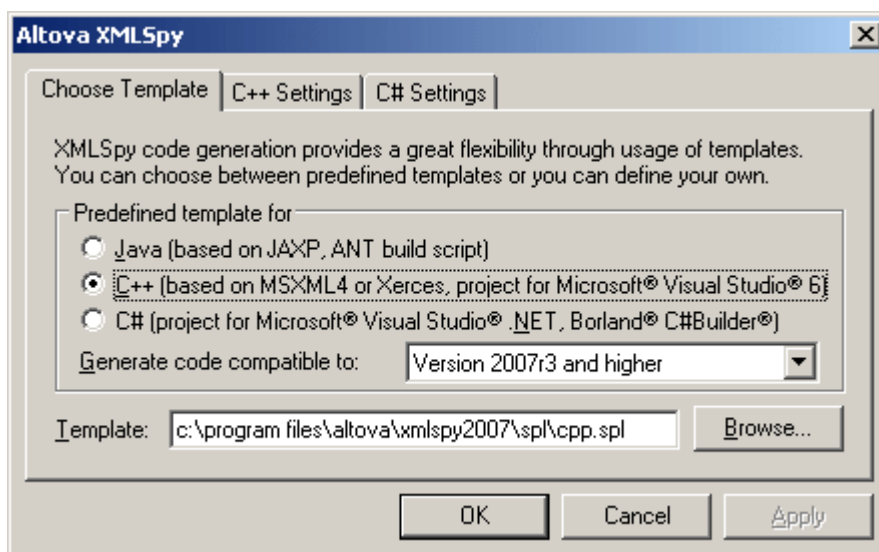
- A random selection.
- Each sample value in turn for each instance of the component. For each file generation, the cycle starts anew.
- The first value always.

Root element

If the schema contains more than one global element, these are listed, and the root element required for the sample XML file can be selected from the list.

21.5.13 Generate Program Code

The **DTD/Schema | Generate Program Code...** command automatically generates Java, C++, or C# class files from definitions in the active schema (DTD or XML Schema) document.



Please see the [Code Generator](#) section of this documentation for details about code generation.

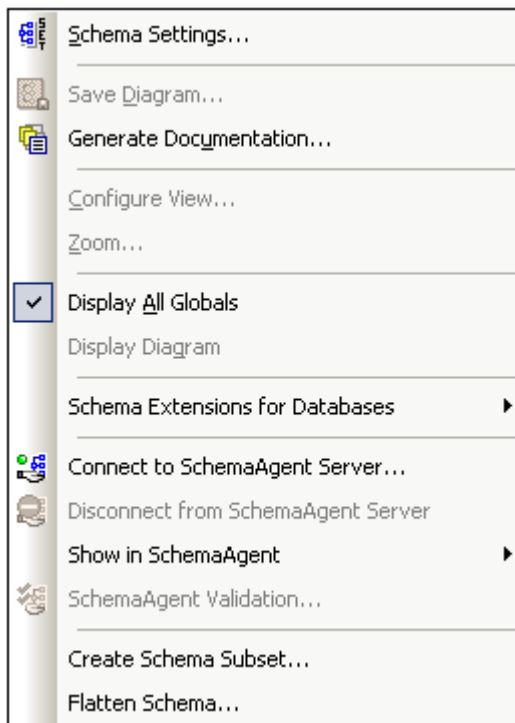
21.5.14 Flush Memory Cache

The **DTD/Schema | Flush Memory Cache** command flushes all cached schema (DTD and XML Schema) documents from memory. To speed up validation and intelligent editing, XMLSpy caches recently used schema documents and external parsed entities in memory. Information from these cached documents is also displayed when the [Go to Definition](#) command is invoked.

Flush the memory cache if memory is tight on your system, or if you have used documents based on different schemas recently.

21.6 Schema Design Menu

The **Schema Design** menu enables you to configure the Schema View of XMLSpy. This view enables you to design XML Schemas in a GUI. It is available when an XML Schema document is active in Schema View.



The commands available in this menu are described in this section.

21.6.1 Schema Settings



The **Schema Design | Schema Settings** command lets you define global settings for the active schema. These settings are attributes and their values of the XML Schema document element, `xs:schema`.

Schema settings

Default Element form: ☒ Qualified ☐ Unqualified
Default Attribute form: ☐ Qualified ☒ Unqualified

Block default:
Final default:
Version:
xml:lang: ID:

☐ No target namespace
☒ Target namespace:

Prefix	Namespace
	http://www.xmlspy.com/schemas/orgchart
xsd	http://www.w3.org/2001/XMLSchema
ipo	http://www.altova.com/IPO
ts	http://www.xmlspy.com/schemas/textstate

OK Cancel

You can make the following settings in this dialog:

- The `elementFormDefault` and `attributeFormDefault` attributes of the `xs:schema` element can each be set to `qualified` or `unqualified`.
- The `blockDefault`, `finalDefault`, and `version` attributes can be entered in their respective fields.
- The target namespace for the XML instance document can be set by selecting the `Target Namespace` radio button, and entering the `target namespace` in the field. If you declare a target namespace, you must define that namespace for use in the schema document. Do this by entering a namespace line in the Namespace list in the bottom pane. You can define a prefix for this namespace, or let this namespace be the default namespace (by leaving the prefix field blank as shown in the screenshot above).

The Text View of the schema settings shown in the screenshot above will look something like this:

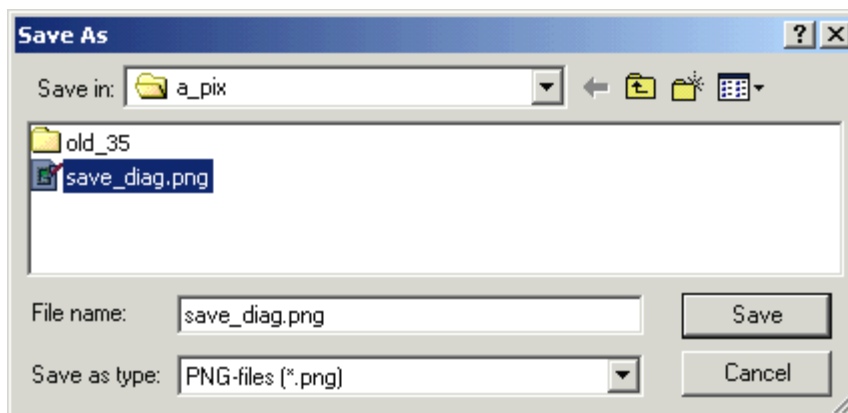
```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
targetNamespace="http://www.xmlspy.com/schemas/Altova/orgchart" xmlns="http://www.xmlspy.com/schemas/Altova/orgchart"
xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
elementFormDefault="unqualified"
attributeFormDefault="unqualified" blockDefault="extension"
finalDefault="restriction" version="3.5a">
  <xsd:notation name="Altova-Orgchart"
public="http://www.xmlspy.com/schemas/Altova/orgchart"/>
  <xsd:complexType name="DivisionType">
    <xsd:sequence>
      <xsd:element name="Name" type="xsd:string">
        <xsd:annotation>
          <xsd:documentation>Division Name
```

Please note: These settings apply to the active schema document only.

21.6.2 Save Diagram



The **Schema Design | Save Diagram...** command saves the diagram of the Content Model currently displayed in the Main Window in PNG format to any desired location.



21.6.3 Generate Documentation



The **Schema Design | Generate Documentation** command generates detailed documentation about your schema (see screenshot below) in HTML, MS Word, RTF or PDF. The documentation generated by this command can be freely altered and used; permission from Altova to do so is not required. Documentation is generated for components you select in the Schema Documentation dialog (which appears when you select the Generate Documentation command). Related elements (child elements, complex types, etc.) are typically hyperlinked in the onscreen output, enabling you to navigate from component to component. Components with a content model also have links to the content model definitions. Note that schema documentation is also generated for **included and imported schema components**. The various documentation-generation options are described in the section, [Documentation Options](#).

Note: In order to generate documentation in MS Word format, you must have MS Word (version 2000 or later) installed.

You can either use XMLSpy's fixed standard design for the generated document, or you can use a StyleVision SPS for the design. Using a StyleVision SPS enables you to customize the design of the generated documentation as well as to generate PDF as an additional output format. How to work with an SPS is explained in the section, [User-Defined Design](#).

Note: In order to use an SPS to generate schema documentation, you must have StyleVision installed on your machine.

Schema **ipo.xsd**

schema location: [C:\Program Files\Altova\XMLSPY2004\Examples\ipo.xsd](#)
 targetNamespace: <http://www.altova.com/IPO>

Elements	Complex types	Simple types
comment	Items	Sku
purchaseOrder	PurchaseOrderType	

schema location: [C:\Program Files\Altova\XMLSPY2004\Examples\address.xsd](#)
 targetNamespace: <http://www.altova.com/IPO>

Complex types	Simple types
Address	EU-Postcode
EU-Address	US-State
US-Address	

element **comment**

diagram	
namespace	http://www.altova.com/IPO
type	string
properties	content simple
used by	element Items/item complexType PurchaseOrderType
source	<code><element name="comment" type="string"/></code>

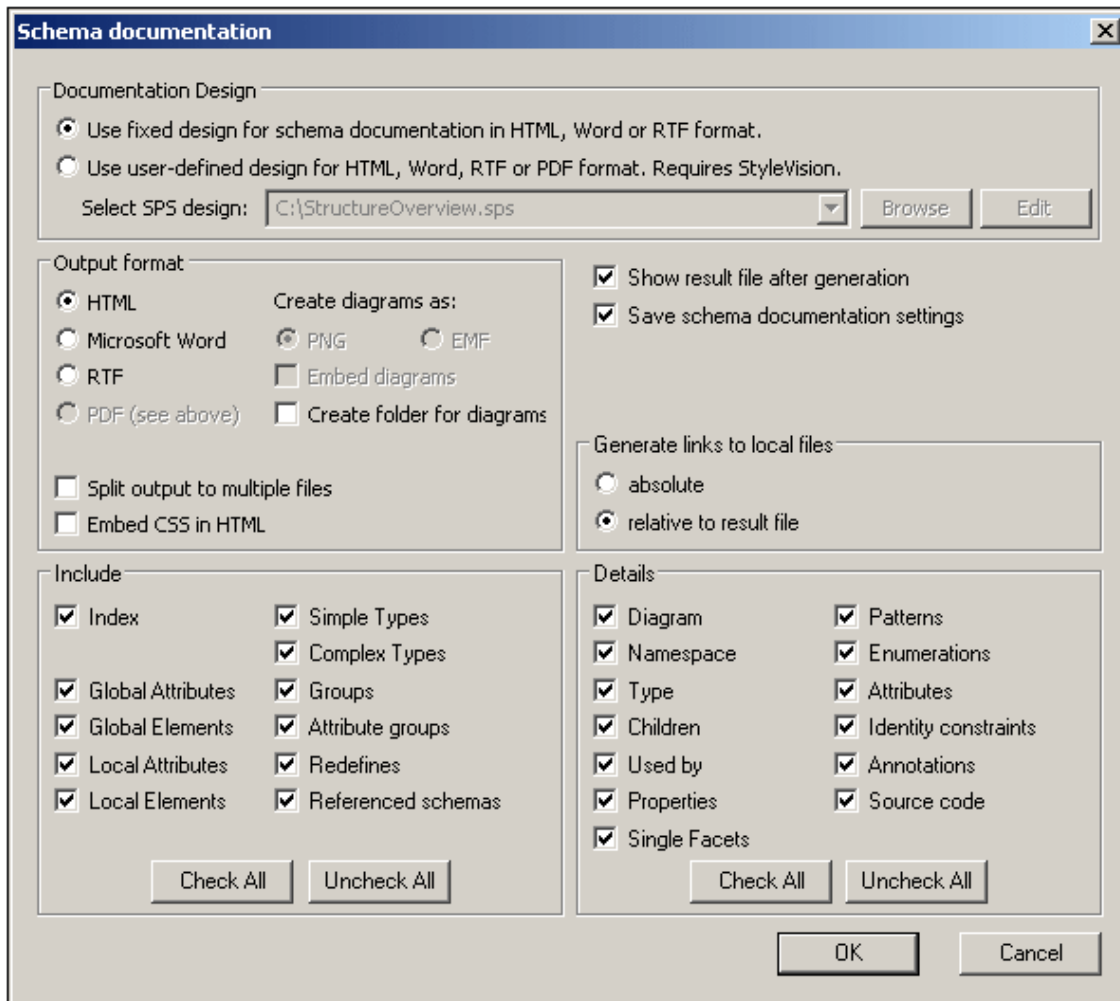
The screenshot above shows generated schema documentation with an index (all related schemas with their global components organized by component type) at the top of the document.

Note: When generating documentation for W3C schema documents, XMLSpy uses application-internal versions of these documents. Consequently, other locations of these documents are not considered, and redefinitions and other schema modifications will not be reflected in the documentation.

Documentation Options

The **Schema Design | Generate Documentation** command pops up the Schema Documentation dialog (*screenshot below*), in which you can select options for the documentation.

In the Documentation Design pane of the dialog you can select whether to use the fixed XMLSpy design for the generated documentation or whether to use a customized design created in a StyleVision SPS. Select the option you want. Note that PDF output is available only for documentation generated with a StyleVision SPS, not for documentation generated using a fixed design. How to work with a user-defined design is described in the section, [User-Defined Design](#).



The other options in the Schema Documentation dialog are explained below:

- The required format is specified in the Output Format pane: either HTML, Microsoft Word, RTF, or PDF. (The PDF output format is only available if you use a StyleVision SPS to generate the documentation.) On clicking **OK**, you will be prompted for the name of the output file and the location to which it should be saved.
- Microsoft Word documents are created with the .doc file extension when generated using a fixed design, and with a .docx file extension when generated using a StyleVision SPS.
- The documentation can be generated either as a single file or be split into multiple files. When multiple files are generated, each file corresponds to a component. What components are included in the output is specified using the check boxes in the Include pane. In fixed designs, links between multiple documents are created automatically.
- For HTML output, the CSS style definitions can be either saved in a separate CSS file

or embedded in the HTML file (in the `<head>` element). If a separate CSS file is created, it will be given the same name as the HTML file, but will have a `.css` extension. Check or uncheck the *Embed CSS in HTML* check box to set the required option.

- The *Embed Diagrams* option is enabled for the MS Word, RTF, and PDF output options. When this option is checked, diagrams are embedded in the result file, either in PNG or EMF format. Otherwise diagrams are created as PNG or EMF files, which are displayed in the result file via object links.
- When the output is HTML, all diagrams are created as document-external PNG files. If the *Create folder for diagrams* check box is checked, then a folder will be created in the same folder as the HTML file, and the PNG files will be saved inside it. This folder will have a name of the format `HTMLFilename_diagrams`. If the *Create folder for diagrams* check box is unchecked, the PNG files will be saved in the same folder as the HTML file.
- Links to local files (such as diagram image files and external CSS file) can be relative or absolute. In the *Generate links to local files* pane, select the appropriate radio button according to the option you prefer.
- In the Include pane, you select which items you want to include in the documentation. The Index option lists all related schemas at the top of the file, with their global components organized by component type. The **Check All** and **Uncheck All** buttons enable you to quickly select or deselect all the options in the pane.
- The Details pane lists the details that may be included for each component. Select the details you wish to include in the documentation. The **Check All** and **Uncheck All** buttons enable you to quickly select or deselect all the options in the pane.
- The *Show Result File* option is enabled for all output options. When this option is checked, the result files are displayed in Browser View (HTML output), MS Word (MS Word output), and the default applications for `.rtf` files (RTF output) and `.pdf` files (PDF output). You can also select whether links in the output document are absolute or relative.

Parameter values

If the StyleVision SPS contains one or more parameter definitions, then on clicking **OK**, a dialog pops up listing all the parameters defined in the SPS. You can enter parameter values in this dialog to override the default parameter values that were assigned in the SPS.

User-Defined Design

Instead of the fixed standard XMLSpy design, you can create a customized design for schema documentation. The customized design is created in a StyleVision SPS, which is a design template for the output document.

Creating the SPS

A StyleVision Power Stylesheet (or SPS) is created using [Altova's StyleVision](#) product. An SPS for generating schema documentation must be based on an XML Schema that specifies the structure of the schema documentation. This schema is called `SchemaDocumentation.xsd`, and it is delivered with your XMLSpy package. It is stored in the folder: `C:\Documents and Settings\<username>\My Documents\Altova\XMLSpy2012\Documentation\Schema`.

When creating the SPS design in StyleVision, nodes from the `SchemaDocumentation.xsd` schema are placed in the design template and assigned styles and properties. Additional components, like links, tables and images, can also be added to the SPS design. In this way, the entire output document can be designed in the SPS. How to create an SPS design in StyleVision is described in detail in the StyleVision user manual.

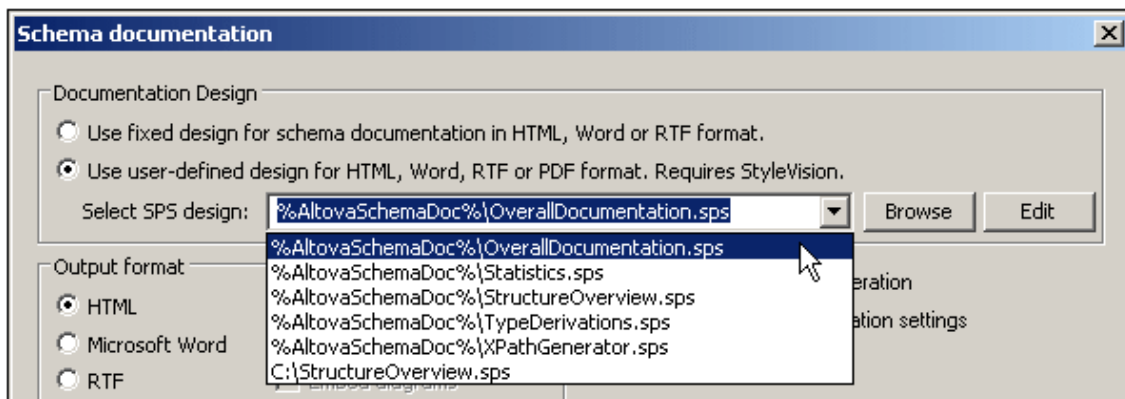
The advantage of using an SPS for generating schema documentation is that you have complete control over the schema documentation design. Note also that PDF output of the

schema documentation is available only if a user-defined SPS is used; PDF output is not available if the fixed XMLSpy design is used.

Specifying the SPS to use for schema documentation

After an SPS has been created, it can be used to generate schema documentation. The SPS you wish to use for generating the schema documentation is selected in the Schema Documentation dialog (accessed via the **Schema Design | Generate Documentation** command). In the Documentation Design pane of this dialog (see *screenshot below*), select the *Use User-Defined Design* radio button. You can then click the **Browse** button and browse for the SPS you want. Click the dialog's **OK** button, and, in the Save dialog that pops up, select the folder for, and enter the name of, the output file.

Note: The SPS file must correctly locate the schema on which it is based:
SchemaDocumentation.xsd (see above).



The following editable SPS designs for schema documentation generation are delivered with XMLSpy. They are in the [\(My Documents\) folder](#): C:\Documents and Settings\

- OverallDocumentation.sps, which generates full documentation about the schema
- Statistics.sps, which lists the number of global and local elements, attributes and attribute groups, and simple and complex types for the main schema and for each schema file independently
- StructureOverview.sps, which outputs a structure of global elements and complex types up to a configurable depth
- TypeDerivations.sps, which lists simple and complex types and all their directly and indirectly derived types in the form of a tree
- XPathGenerator.sps, which generates all possible XPath statements up to a configurable depth

These files, together with other SPS files you have recently browsed for, will be available in the combo box of the *Use User-Defined* option (see *screenshot above*).

Clicking the **Edit** button in the Documentation Design pane launches StyleVision and opens the selected SPS in a StyleVision window. In order to preview the result document in StyleVision, you will need a Working XML file. A sample XML file for this purpose, called OrgChart.xml, is supplied with your application and is located in the [\(My Documents\) folder](#):

C:\Documents and Settings\

Note: In order to use an SPS to generate schema documentation, you must have StyleVision

installed on your machine.

21.6.4 Configure View

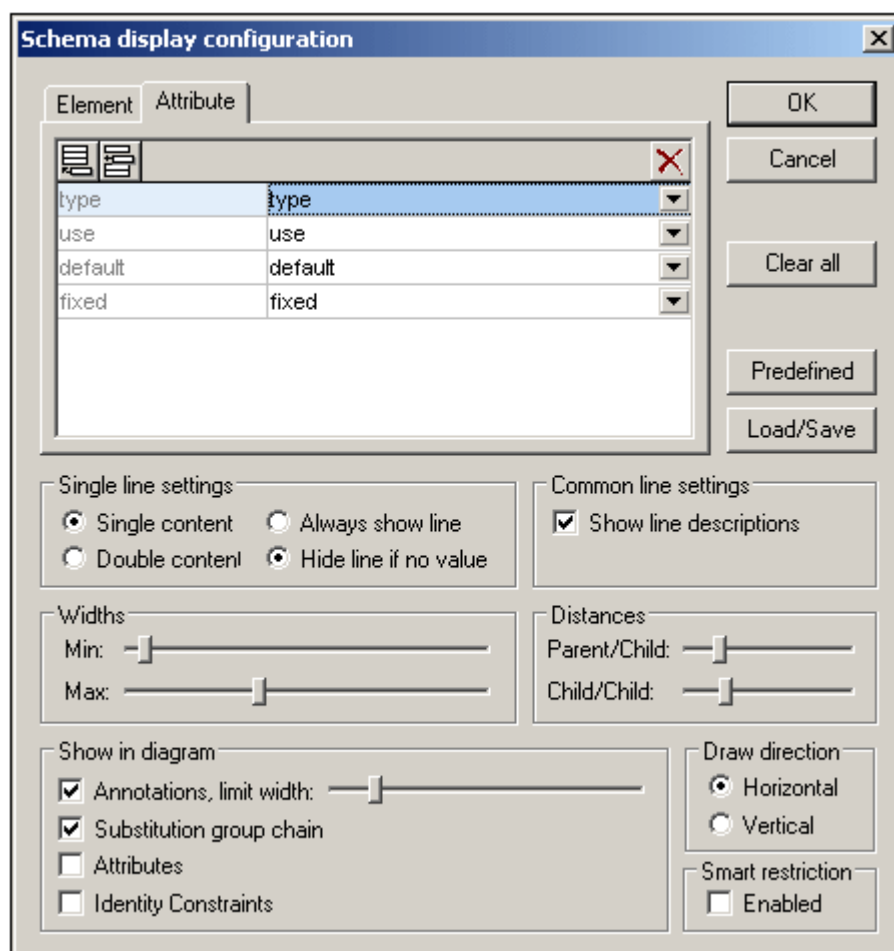
The **Schema Design | Configure view** command is active in Content Model View and allows you to configure the Content Model View. Clicking the command opens the Schema Display Configuration dialog at the bottom right of the XMLSpy window, enabling you to see the effect of your settings as you enter them in the dialog. The settings take effect when you click the **OK** button of the dialog, and apply to the Content Model View of all XML Schema files that are opened subsequently. These settings also apply to the schema documentation output and printer output.

Defining property descriptor lines for the content model

You can define what properties of elements and attributes are displayed in the Content Model View. These properties appear as grid lines in component boxes.

To define property descriptor lines:

1. Select **Schema Design | Configure view**. The Schema display configuration dialog appears.
2. In the **Element** or **Attribute** tab, click the Append  or Insert  icon to add a property descriptor line. The line is added in the dialog and to element boxes in the Content Model View.
3. From the combo box, select the property you want to display. *See screenshot.*
4. Repeat steps 2 and 3 for as many properties as required.



The Content Model View is updated, showing the defined property descriptor lines for all elements for which they exist.

Please note:

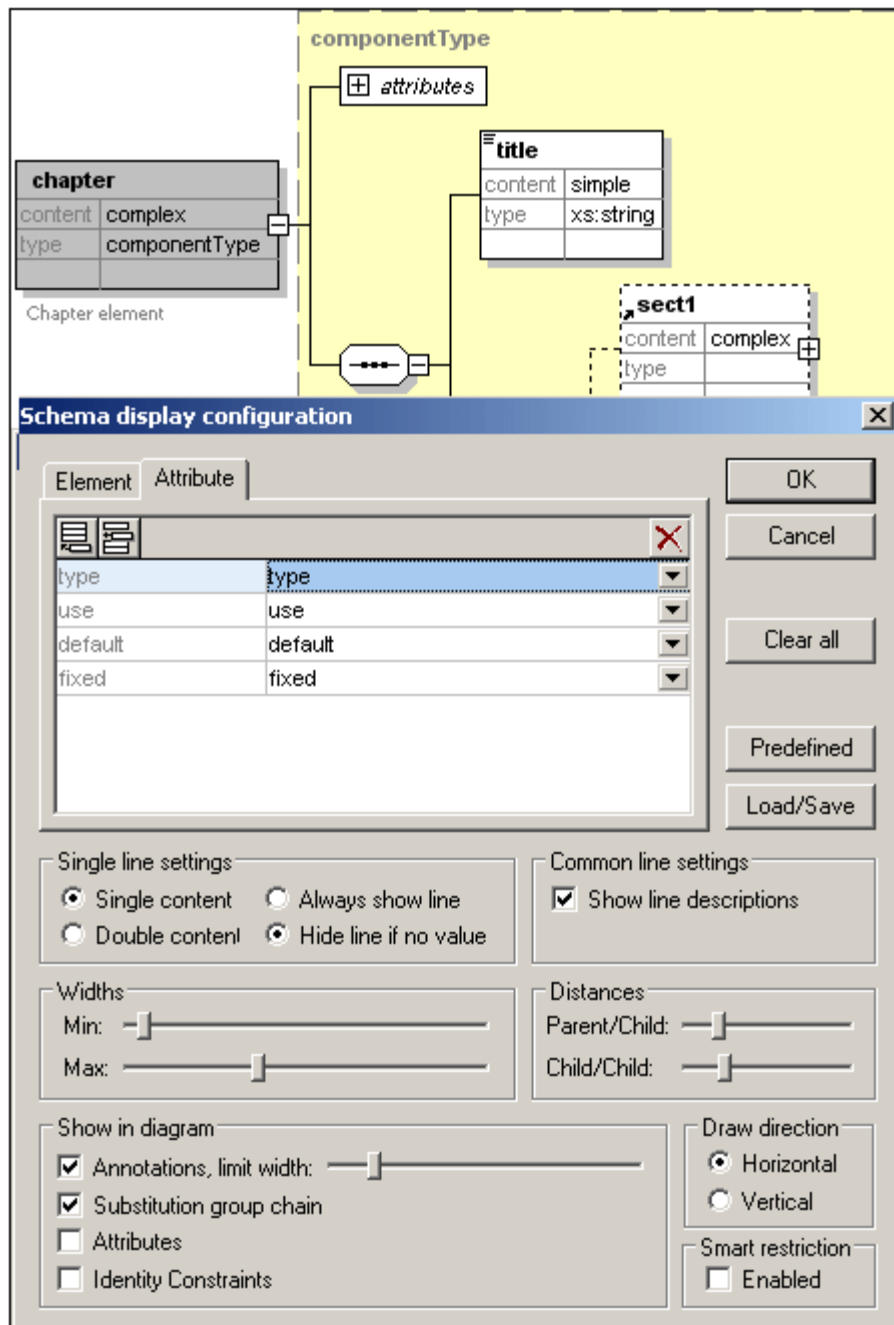
- For attributes, the configuration you define appears only when attributes are displayed in the diagram (as opposed to them being displayed in a pane below the Content Model View).
- The configured view applies to all Content Model Views opened after the configuration is defined.

Deleting a property descriptor line from the Content Model View

To delete individual property descriptor lines, in the Schema Display Configuration dialog, select the property descriptor line you want to delete, and click the Delete icon .

Settings for configuring the Content Model View

The Content Model View can be configured using settings in the Schema Display Configuration dialog. How to define what property descriptor lines are displayed in Content Model View has been described above. The other settings are described below.



Single line settings

You can define whether a property descriptor line is to contain single or double content, and whether individual lines must appear for every element or only for elements that contain that property. Use the appropriate radio buttons to define your settings. Note that these two settings can be set for individual lines separately (select the required line and make the setting).

Common line settings

This option toggles the line descriptions (i.e. the name of the property) on and off.

Widths

These sliders enable you to set the minimum and maximum size of the element rectangles in Content Model View. Change the sizes if line descriptor text is not fully visible or if you want to standardize your display.

Distances

These sliders let you define the horizontal and vertical distances between various elements onscreen.

Show in diagram

The Annotations check box toggles the display of annotation text on or off, as well as the annotation text width with the slider. You can also toggle the display of the substitution groups on or off. The Attributes and Identity Constraints appear in the Content Model diagram if their check boxes are selected; otherwise they appear as tabs in a pane at the bottom of the Content Model window.

Draw direction

These options define the orientation of the element tree on screen, horizontal or vertical.

Editing the content model in the diagram itself

You can change element properties directly in the content model diagram. To do this, double-click the property you wish to edit and start entering data. If a selection is available, a drop-down list appears, from which you can select an option. Otherwise, enter a value and confirm with **Enter**.

Buttons in the Schema display configuration dialog

This dialog has the following buttons:

- The **Load/Save** button allows you to load and save the settings you make here.
- The **Predefined** button, resets the display configuration to default values.
- The **Clear all** button empties the list box of all entries.

Enabling smart restrictions

To enable [smart restrictions](#), check the Enable Schema Restrictions check box.

21.6.5 Zoom

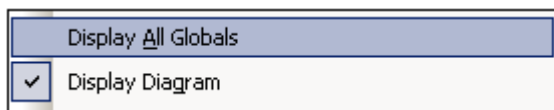
The **Schema Design | Zoom** command controls the zoom factor of the Content Model View. This feature is useful if you have a large content model and wish to zoom out so that the entire content model fits in the Main Window. You can zoom between 10% and 200% of actual size.



To zoom in and out, either drag the slider or click in the entry box and enter a percentage value.

21.6.6 Display All Globals

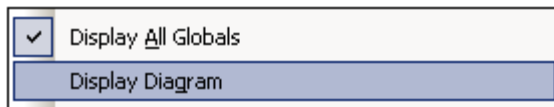
The **Schema Design | Display All Globals** command switches from [Content Model View](#) to [Schema Overview](#) to display all global components in the schema. It is a toggle with the Display Diagram command. The currently selected toggle is indicated with a check mark to its left (see *screenshot*).



Alternatively, you could use the **Display All Globals** icon  at the top of the Content Model View to switch to the Schema Overview.

21.6.7 Display Diagram

The **Schema Design | Display Diagram** command switches to the [Content Model View](#) of the selected global component—if the selected component has a content model. Global components that have a content model (complex types, elements, and element groups) are indicated with the  icon to its left. The Display Diagram command is a toggle with the Display All Globals command. The currently selected toggle is indicated with a check mark to its left (see *screenshot below*).



Alternatively, you could use the following methods to switch to Content Model View:

- Click the  icon next to the component, the content model of which you want to display.
- Double-click a component name in the Component Navigator Entry Helper (at top right).

21.6.8 Schema Extensions for Databases

This menu item pops out a sub-menu containing commands for Oracle and MS SQL Server schema extensions.

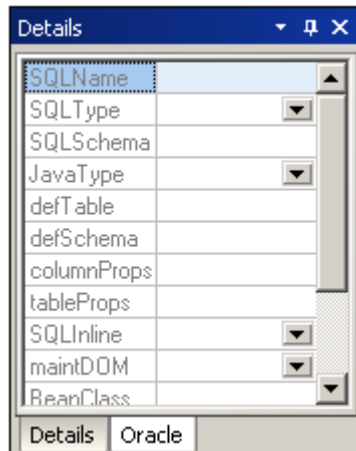
- [Enable Oracle Schema Extensions](#)
- [Oracle Schema Settings](#)
- [Enable Microsoft SQL Server Schema Extensions](#)
- [Named Schema Relationships](#)
- [Unnamed Element Relationships](#)

Enable Oracle Schema Extensions

XMLSpy provides support for Oracle schema extensions for use with Oracle 9i Project XDB. Using these schema extensions allows you to configure and customize how Oracle 9i Project XDB stores XML documents. These XML documents are then accessible through SQL queries and legacy tools. Please see the [Oracle Website](#) for more information.

When you select the **Schema Design | Enable Oracle Schema Extensions** command, the following occurs:

- The XDB namespace is declared on the `schema` element:
`xmlns:xdb="http://xmlns.oracle.com/xdb"`.
- An Oracle tab is created in the Details Entry Helper, enabling you to add attributes—including XDB-specific attributes—to schema elements such as `xsd:complexType` and `xsd:element`.

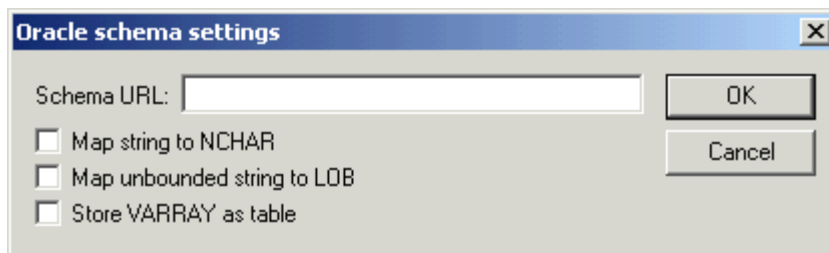


Oracle extensions can be defined for complex types, elements, and attributes. Use the Entry Helper as you normally would in XMLSpy.

Please note: This menu command can be toggled on and off, that is, extensions can be enabled or disabled. When Oracle extensions are enabled, the command is displayed with a check mark to its left. Disabling Oracle extensions (by clicking the enabled command) deletes the XDB namespace declaration and all XDB extensions in the file. A warning message appears since this action cannot be undone.

Oracle Schema Settings

The **Schema Design | Oracle Schema Settings** command allows you to define global settings for Oracle schema extensions.



In order to access this dialog, Oracle schema extensions must be enabled (using the [Enable Oracle Schema Extensions](#) command).

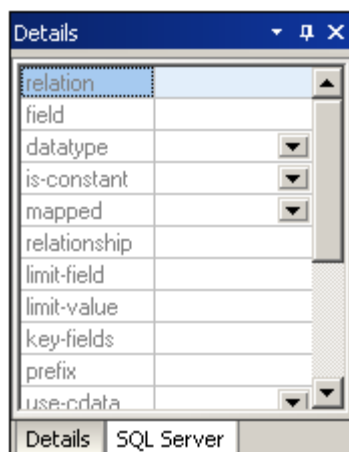
Enable Microsoft SQL Server Schema Extensions

XMLSpy provides support for Microsoft SQL Server 2000 schema extensions for use with Microsoft SQL Server. Using these schema extensions allows you to configure and customize how Microsoft SQL Server stores XML documents. These XML documents are then accessible

through SQL queries and legacy tools. Please see the [Microsoft Website](#) for more information.

When you select the **Schema Design | Enable Microsoft SQL Server Schema Extensions** command, the following occurs:

- The SQL Server namespace is declared on the `schema` element:
`xmlns:sql="urn:schemas-microsoft-com:mapping-schema"`.
- An SQL Server tab is created in the Details Entry Helper, enabling you to add attributes to schema elements such as `xsd:element`.



Where SQL Server extensions can be defined for a schema component, the SQL Server tab is available in the Details Entry Helper when the component is selected. Use the Entry Helper as you normally would in XMLSpy.

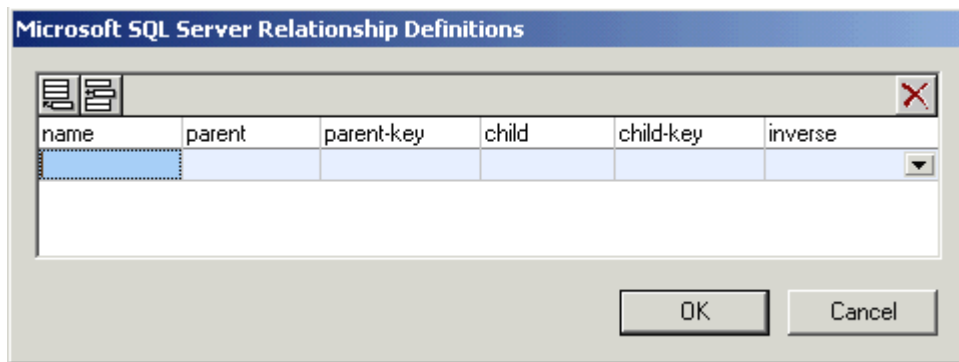
Please note: This menu command can be toggled on and off, that is, extensions can be enabled or disabled. When SQL Server extensions are enabled, the command is displayed with a check mark to its left. Disabling SQL Server extensions (by clicking the enabled command) deletes the SQL Server namespace declaration and all SQL extensions in the file. A warning message appears since this action cannot be undone.

Named Schema Relationships

The **Schema Design | Named Schema Relationships** command allows the definition of named relationships to provide the information needed to create the document hierarchy. You have to have previously enabled the SQL Server schema extensions, using the menu option "Enable SQL Server Schema Extensions", to be able to access this menu option.

To create a named schema relationship:

1. Click the insert  or append icon , to add a new row to the dialog box.
2. Click the field and enter the corresponding relationship name.
3. Click **OK** to confirm.



This generates a SQL relationship element, placing it just after the namespace declaration.

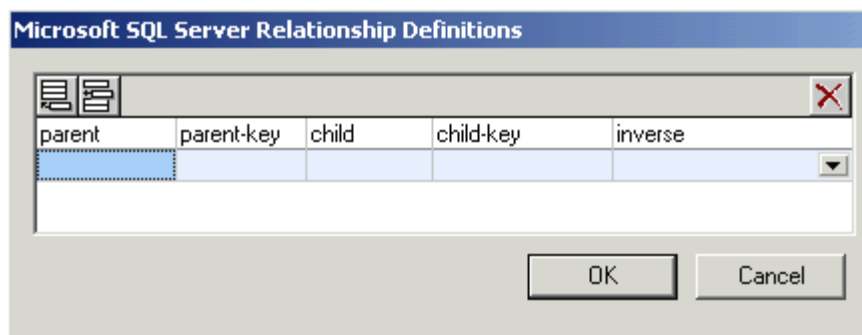
Please note: Click the delete icon , to delete a row from the dialog box.

Unnamed Element Relationships

The **Schema Design | Unnamed Element Relationships** command allows the definition of unnamed relationships to provide the information needed to create the document hierarchy. You have to have previously enabled the SQL Server schema extensions, using the menu option **Enable Microsoft SQL Server Schema Extensions**, to be able to access this menu option.

To create an unnamed schema relationship:

1. Click the insert  or append icon , to add a new row to the dialog box.
2. Click the field and enter the corresponding relationship name.
3. Click **OK** to confirm.



This generates a SQL relationship element for the currently selected schema element.

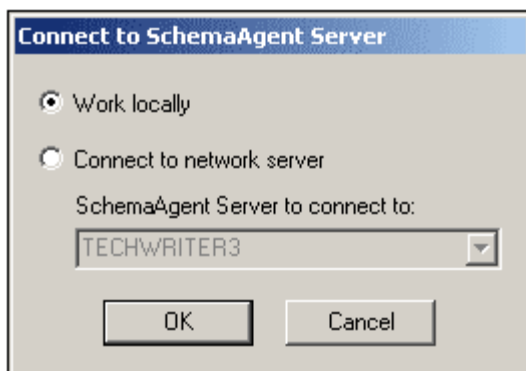
Please note: Click the delete icon , to delete a row from the dialog box.

21.6.9 Connect to SchemaAgent Server



The **Schema Design | Connect to SchemaAgent Server** command is enabled when an XML Schema document is active and it enables you to connect to a SchemaAgent Server. You are

able to connect to a SchemaAgent server only if a licensed Altova SchemaAgent product is installed on your machine. When you click this command, the Connect to SchemaAgent Server dialog (*screenshot below*) opens:



You can use either the local server (the SchemaAgent server that is packaged with Altova SchemaAgent) or a network server (the Altova SchemaAgent Server product, which is available free of charge). If you select **Work Locally**, the local server of SchemaAgent will be started when you click **OK** and a connection with it will be established. If you select **Connect to Network Server**, the selected SchemaAgent Server must be running in order for a connection to be made.

When connected to SchemaAgent Server, XMLSpy acts as a SchemaAgent client, and provides powerful and enhanced schema editing and management functionality. For details about SchemaAgent, the installation of SchemaAgent Server, and how to connect to SchemaAgent Server, see [SchemaAgent](#) in the DTD and XML Schema section of this user manual. For more information about installing and working with these two products, see the SchemaAgent user manual that is delivered with these products.

After you connect to SchemaAgent Server, a message appears in the bar at the top of the Main Window with information about the connection. You now have full access to all schemas and schema components in the search path/s (folder/s) defined for the SchemaAgent server to which XMLSpy is connected.

Please note: In order for the connection to succeed, you must have Altova's SchemaAgent Client product installed with a valid license on the same machine as that on which XMLSpy is installed.

21.6.10 Disconnect from SchemaAgent Server



The **Disconnect from SchemaAgent Server** command is enabled when a connection to a SchemaAgent Server has been made successfully. Selecting this command disconnects XMLSpy from the SchemaAgent Server.

21.6.11 Show in SchemaAgent

The **Show in SchemaAgent** menu item causes the active schema and, optionally, linked schemas to be displayed in the Altova product SchemaAgent. (This product must be installed on the same machine as XMLSpy if you wish to use SchemaAgent functionality). The schema/s are opened in a new SchemaAgent Design in SchemaAgent.

Mousing over the **Show in SchemaAgent** menu item pops out a submenu with options about what schemas to show in SchemaAgent. These options are described in [SchemaAgent](#) in the DTD and XML Schema section of this user manual.

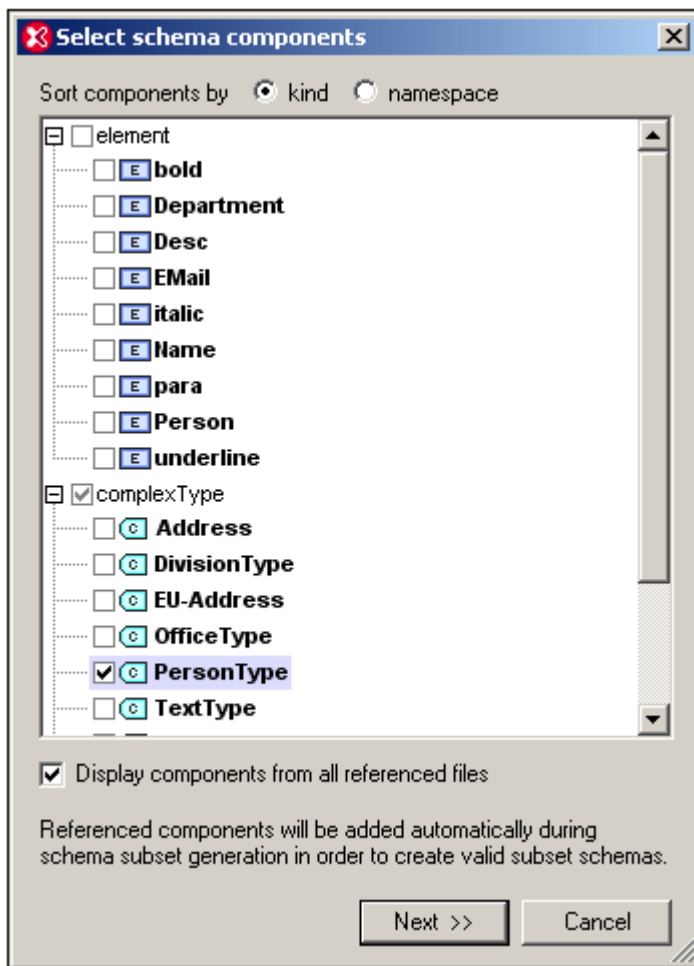
21.6.12 SchemaAgent Validation



The **SchemaAgent Validation** command enables you to validate the currently active schema as well as schemas related to the currently active schema. This feature is described in detail in the [SchemaAgent Validation](#) section in the Schema View section of this user manual.

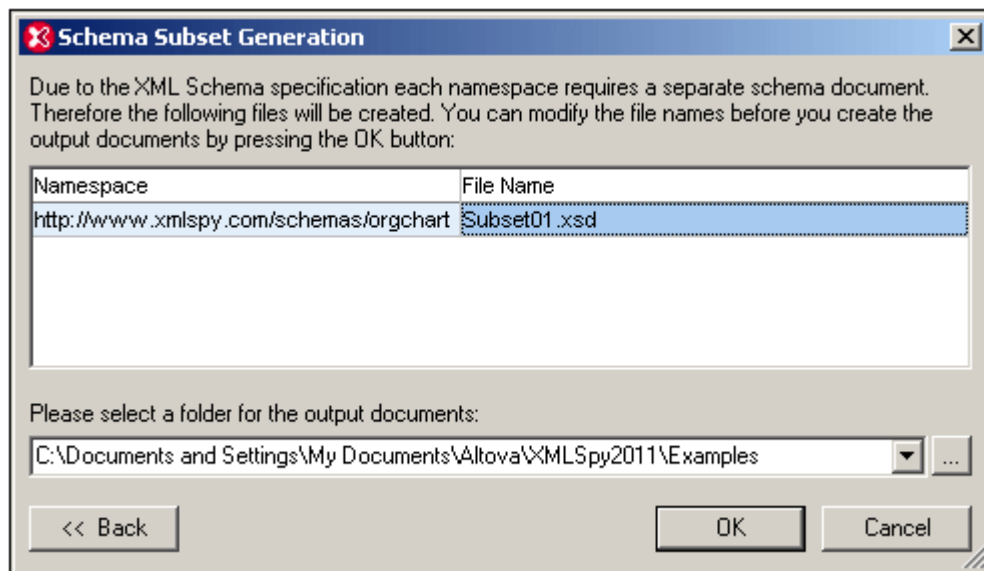
21.6.13 Create Schema Subset

The **Create Schema Subset** command pops up the Select Schema Components dialog (*screenshot below*). In this dialog, you check the component or components you wish to create as a single schema subset, then click **Next**. (Note that a check box below the pane enables components from all referenced files to also be listed for selection.)



In the Schema Subset Generation dialog that now appears (*screenshot below*), enter the name/

s you want the file/s of the schema subset package to have. You must also specify the folder in which the new schema subset files are to be saved. A schema subset package could have multiple files if one or more of the components being created is an imported component in the original schema. A separate schema file is created for each namespace in the schema subset. The filenames displayed in the dialog are, by default, the names of the original files. But since you are not allowed to overwrite the original files, use new filenames if you wish to save the files in the same folder as the original files.

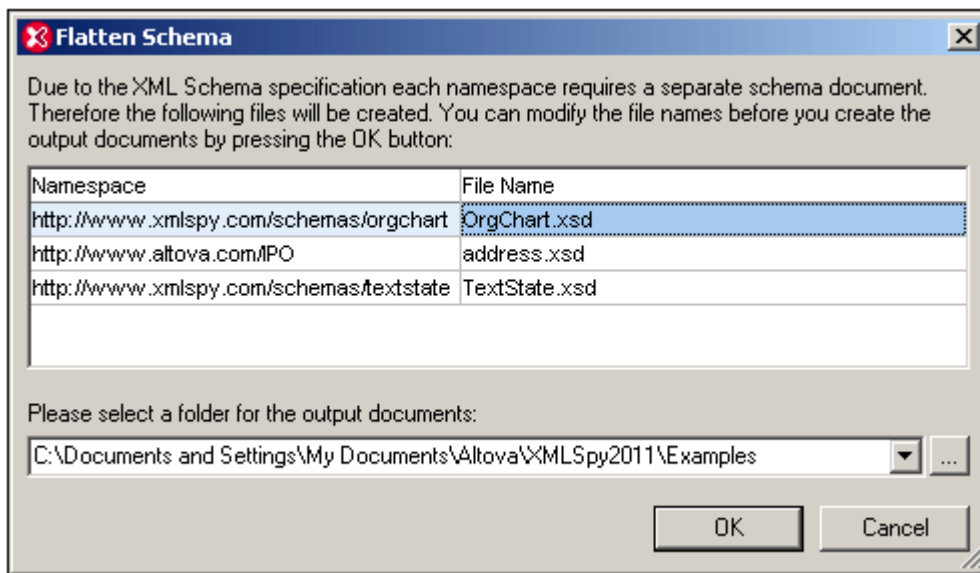


On clicking **OK**, the schema subset file with the namespace corresponding to that of the active file is opened in Schema View. Any other files in the package are created but not opened in Schema View.

21.6.14 Flatten Schema

Flattening the active schema in Schema View is the process of: (i) adding the components of all included schemas as global components of the active schema, and (ii) deleting the included schemas.

To flatten the active schema, select the command **Schema Design | Flatten Schema**. This pops up the Flatten Schema dialog (*screenshot below*), which contains the names of separate files, one for each namespace that will be in the flattened schema. These default names are the same as the original filenames. But since you are not allowed to overwrite the original files, the filenames must be changed if you wish to save in the same folder as the active file. You can browse for a folder in which the flattened schema and its associated files will be saved.



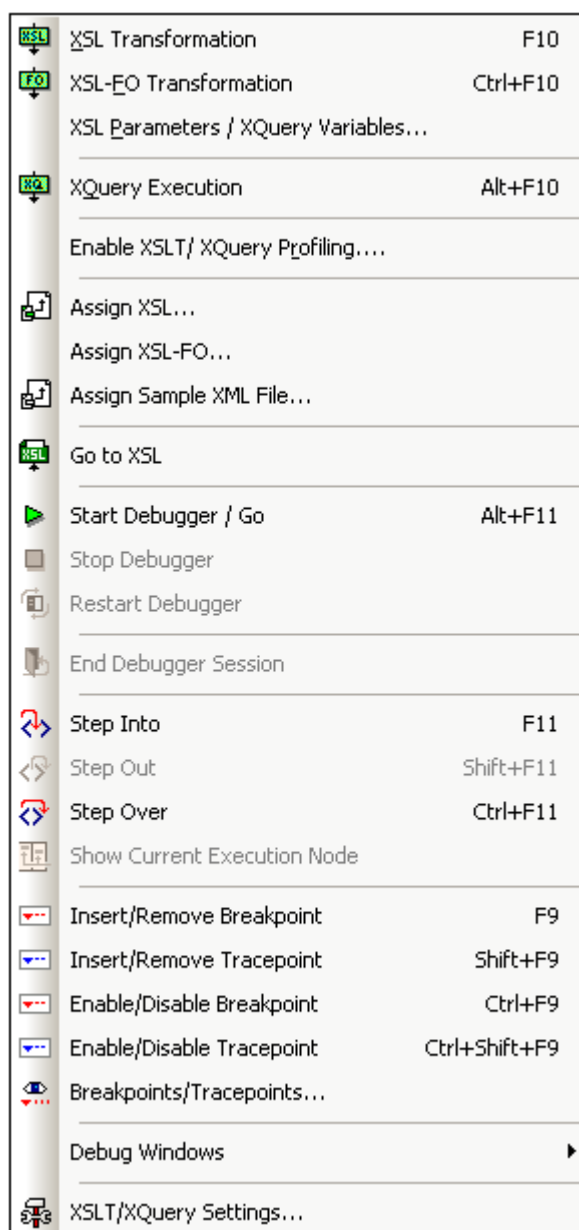
On clicking **OK**, the flattened schema file will be opened in Schema View.

21.7 XSL/XQuery Menu

The XSL Transformation language lets you specify how an XML document should be converted into other XML documents or text files. One kind of XML document that is generated with an XSLT document is an FO document, which can then be further processed to generate PDF output. XMLSpy contains built-in XSLT processors (for XSLT 1.0 and XSLT 2.0) and can link to an FO processor on your system to transform XML files and generate various kinds of outputs. The location of the FO processor must be specified in the XSL tab of the Options dialog ([Tools | Options](#)) in order to be able to use it directly from within the XMLSpy interface.

XMLSpy also has a built-in XQuery engine, which can be used to execute XQuery documents (with or without reference to an XML document).

Commands to deal with all the above transformations are accessible in the **XSL/XQuery** menu. In addition, this menu also contains commands to work with the Altova XSLT/XQuery Debugger.



21.7.1 XSL Transformation



F10

The **XSL/XQuery | XSL Transformation** command transforms an XML document using an assigned XSLT stylesheet. The transformation can be carried out using the appropriate built-in Altova XSLT Engine (Altova XSLT 1.0 Engine for XSLT 1.0 stylesheets; Altova XSLT 2.0 Engine for XSLT 2.0 stylesheets), the Microsoft-supplied MSXML module, or an external XSLT processor. The processor that is used in conjunction with this command is specified in the [XSL tab](#) of the Options dialog (**Tools | Options**).

If your XML document contains a reference to an XSLT stylesheet, then this stylesheet is used

for the transformation. (An XSLT stylesheet can be assigned to an XML document using the [Assign XSL](#) command. If the XML document is part of a project, an XSLT stylesheet can be specified on a per-folder basis in the [Project Properties](#) dialog. Right-click the project folder/s or file/s you wish to transform and select XSL Transformation.) If an XSLT stylesheet has not been assigned to an XML file, you are prompted for the XSLT stylesheet to use. You can also select a file via a global resource or a URL (click the [Browse](#) button) or a file in one of the open windows in XMLSpy (click the **Window** button).

Automating XSLT transformations with AltovaXML 2012

AltovaXML is a free application which contains Altova's XML Validator, XSLT 1.0, XSLT 2.0, and XQuery 1.0 engines. It can be used from the command line, via a COM interface, in Java programs, and in .NET applications to validate XML documents, transform XML documents using XSLT 1.0 and 2.0 stylesheets, and execute XQuery documents.

XSLT transformation tasks can therefore be automated with the use of AltovaXML. For example, you can create a batch file that calls AltovaXML to transform a set of documents. See the [AltovaXML documentation](#) for details.

Transformations to ZIP files

In order to enforce output to a ZIP file, including Open Office XML (OOXML) files such as .docx, one must specify the ZIP protocol in the file path of the output file. For example:

```
filename.zip| zip/filename.xxx
filename.docx| zip/filename.xxx
```

Note: The directory structure might need to be created before running the transformation. If you are generating files for an Open Office XML archive, you would need to zip the archive files in order to create the top-level OOXML file (for example, .docx).

21.7.2 XSL-FO Transformation



Ctrl+F10

FO is an XML format that describes paged documents. An FO processor, such as the Apache XML Project's FOP, takes an FO file as input and generates PDF as output. So, the production of a PDF document from an XML document is a two-step process.

1. The XML document is transformed to an FO document using an XSLT (aka XSL-FO) stylesheet.
2. The FO document is processed by an FO processor to generate PDF (or some alternative output).

The **XSL/XQuery | XSL:FO Transformation** command transforms an XML document or an FO document to PDF.

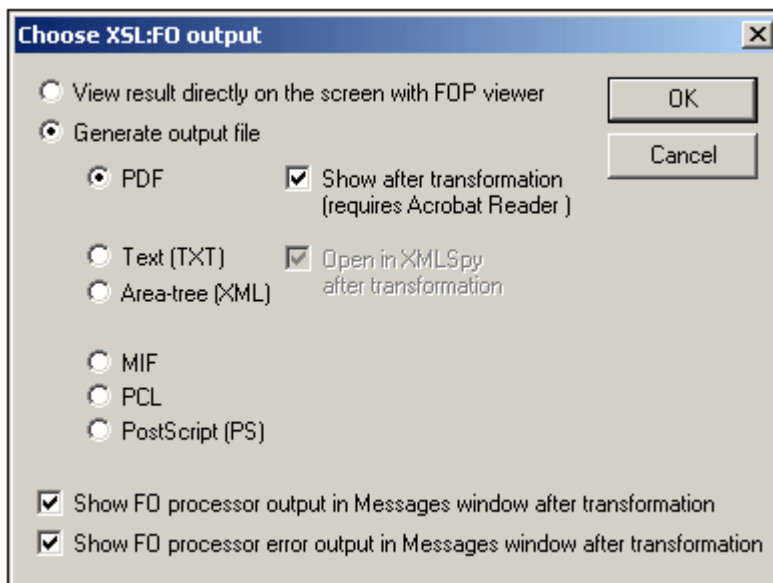
- If the **XSL:FO Transformation** command is executed on a source XML document, then both of the steps listed above are executed, in sequence, one after the other. If the XSLT (or XSL-FO) stylesheet required to transform to FO is not referenced in the XML document, you are prompted to assign one for the transformation (*screenshot below*). Note that you can also select a file via a global resource or a URL (click the [Browse](#) button) or a file in one of the open windows in XMLSpy (click the **Window** button). The transformation from XML to XSL-FO is carried out by the XSLT processor specified in the [XSL tab](#) of the Options dialog (**Tools | Options**). By default the selected XSLT processor is XMLSpy's built-in XSLT processor. The resultant FO document is directly

processed with the FO processor specified in the [XSL tab](#) of the Options dialog (**Tools | Options**).

- If the **XSL:FO Transformation** command is executed on an FO document, then the document is processed with the FO processor specified in the [XSL tab](#) of the Options dialog (**Tools | Options**).

XSL:FO Transformation output

The **XSL:FO Transformation** command pops up the Choose XSL:FO Output dialog (*screenshot below*). (If the active document is an XML document without an XSLT assignment, you are first prompted for an XSLT file.)



You can view the output of the FO processor directly on screen using FOP viewer or you can generate an output file in any one of the following formats: PDF, text, an XML area tree, MIF, PCL, or PostScript. You can also switch on messages from the FO processor to show (i) the processor's standard output message in the Messages window; and (ii) the processor's error messages in the Messages window. To switch on either these two options, check the appropriate check box at the bottom of the dialog.

Please note:

- The Apache FOP processor can be downloaded free of charge using the link at the [Altova Download Center](#). After downloading and installing FOP, you must set the path to the FOP batch file in the [XSL tab](#) of the Options dialog (**Tools | Options**).
- The XSL:FO Transformation command can not only be used on the active file in the Main Window but also on any file or folder you select in the active project. To do this, right-click and select **XSL:FO Transformation**. The XSLT stylesheet assigned to the selected project folder is used.

21.7.3 XSL Parameters / XQuery Variables

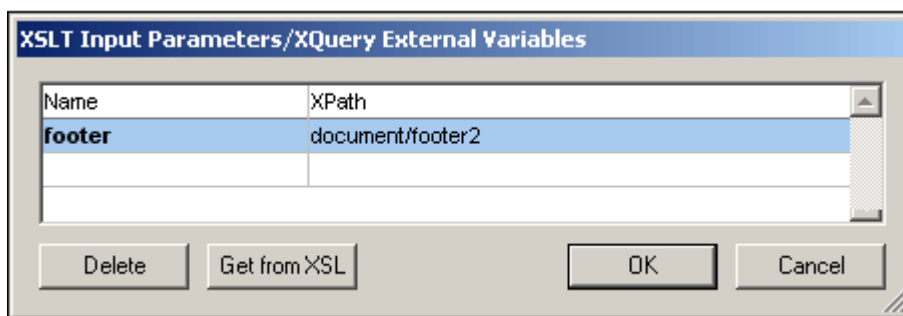
The **XSL/XQuery | XSL Parameters/XQuery Variables** command opens the XSLT Input Parameters/XQuery External Variables dialog (see *screenshot*). You can enter the name of one or more parameters you wish to pass to the XSLT stylesheet, or one or more external XQuery variables you wish to pass to the XQuery document, and their respective values. These parameters are used as follows in XMLSpy:

- When the **XSL Transformation** command in the XSL/XQuery menu is used to transform an XML document, the parameter values currently saved in the dialog are passed to the selected XSLT document and used for the transformation.
- When the **XQuery Execution** command in the XSL/XQuery menu is used to process an XQuery document, the XQuery external variable values currently saved in the dialog are passed to the XQuery document for the execution.

Please note: Parameters or variables that you enter in the XSLT Input Parameters/XQuery External Variables dialog are only passed on to the built-in Altova XSLT engine. Therefore, if you are using MSXML or another external engine that you have configured, these parameters are not passed to this engine.

Using XSLT Parameters

The value you enter for the parameter can be an XPath expression without quotes or a text string delimited by quotes. If the active document is an XSLT document, the **Get from XSL** button will be enabled. Clicking this button inserts parameters declared in the XSLT into the dialog together with their default values. This enables you to quickly include declared parameters and then change their default values as required.



Please note: Once a set of parameter-values is entered in the XSLT Input Parameters/XQuery External Variables dialog, it is used for all subsequent transformations until it is explicitly deleted or the application is restarted. Parameters entered in the XSLT Input Parameters/XQuery External Variables dialog are specified at the application-level, and will be passed to the respective XSLT document for every transformation that is carried out via the IDE from that point onward. This means that:

- parameters are not associated with any particular document
- any parameter entered in the XSLT Input Parameters/XQuery External Variables dialog is erased once XMLSpy has been closed.

Usage example for XSLT parameters

In the following example, we select the required document footer from among three possibilities in the XML document (footer1, footer2, footer3).

```
<?xml version="1.0" encoding="UTF-8"?>
<document xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="C:\workarea\footers\footers.xsd">
  <footer1>Footer 1</footer1>
  <footer2>Footer 2</footer2>
  <footer3>Footer 3</footer3>
  <title>Document Title</title>
  <para>Paragraph text.</para>
  <para>Paragraph text.</para>
</document>
```


The XSLT file contains a local parameter called `footer` in the template for the root element. This parameter has a default value of `footer1`. The parameter value is instantiated subsequently in the template with a `$footer` value in the definition of the footer block.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fo="http://www.w3.org/1999/XSL/Format">
  ...
  <xsl:param name="footer" select="document/footer1" />
  ...
  <xsl:template match="/">
    <fo:root>
      <xsl:copy-of select="$fo:layout-master-set" />
      <fo:page-sequence master-reference="default-page"
        initial-page-number="1" format="1">
        <fo:static-content flow-name="xsl-region-after"
          display-align="after">
          ...
          <fo:inline color="#800000" font-size="10pt" font-weight="bold">
            <xsl:value-of select="$footer"/>
          </fo:inline>
          ...
        </fo:static-content>
      </fo:page-sequence>
    </fo:root>
  </xsl:template>
</xsl:stylesheet>
```

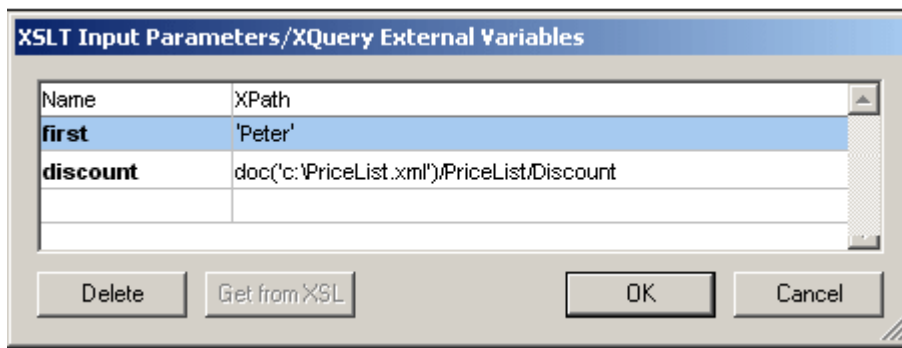
In the XSLT Input Parameters dialog, a new value for the `footer` parameter can be entered, such as the XPath: `document/footer2` (see *screenshot above*) or a text string. During transformation, this value is passed to the `footer` parameter in the template for the root element and is the value used when the footer block is instantiated.

Note:

- If you use the **XSL:FO Transformation** command (**XSL/XQuery | XSL:FO Transformation**), parameters entered in the XSLT Input Parameters/XQuery External Variables dialog are **not** passed to the stylesheet. In order for these parameters to be used in PDF output, first transform from XML to FO using the XSLT Transformation command (**XSL/XQuery | XSL Transformation**), and then transform the FO to PDF using the **XSL:FO Transformation** command (**XSL/XQuery | XSL:FO Transformation**).
- If you use an XSLT processor other than the built-in Altova XSLT Engines, parameters you enter using the Input Parameters dialog will not be passed to the external processor.

Using external XQuery variables

The value you enter for an external XQuery variable could be an XPath expression without quotes or a text string delimited by quotes. The datatype of the external variable is specified in the variable declaration in the XQuery document.



Note: Once a set of external XQuery variables are entered in the XSLT Input Parameters/XQuery External Variables dialog, they are used for all subsequent executions until they are explicitly deleted or the application is restarted. Variables entered in the XSLT Input Parameters/XQuery External Variables dialog are specified at the application-level, and will be passed to the respective XQuery document for every execution that is carried out via the IDE from that point onward. This means that:

- Variables are not associated with any particular document
- Any variable entered in the XSLT Input Parameters/XQuery External Variables dialog is erased once the application (XMLSpy) has been closed down.

Usage example for external XQuery variables

In the following example, a variable `$first` is declared in the XQuery document and is then used in the return clause of the FLWOR expression:

```
xquery version "1.0";
declare variable $first as xs:string external;
let $last := "Jones"
return concat($first, " ", $last )
```

This XQuery returns `Peter Jones`, if the value of the external variable (entered in the XSLT Input Parameters/XQuery External Variables dialog) is `Peter`. Note the following:

- The `external` keyword in the variable declaration in the XQuery document indicates that this variable is an external variable.
- Defining the static type of the variable is optional. If a datatype for the variable is not specified in the variable declaration, then the variable value is assigned the type `xs:untypedAtomic`.
- If an external variable is declared in the XQuery document, but no external variable of that name is passed to the XQuery document, then an error is reported.
- If an external variable is declared and is entered in the XSLT Input Parameters/XQuery External Variables dialog, then it is considered to be in scope for the XQuery document being executed. If a new variable with that name is declared within the XQuery document, the new variable temporarily overrides the in-scope external variable. For example, the XQuery document below returns `Paul Jones` even though the in-scope external variable `$first` has a value of `Peter`.

```
xquery version "1.0";
declare variable $first as xs:string external;
let $first := "Paul"
let $last := "Jones"
return concat($first, " ", $last )
```

Note: It is not an error if an external XQuery variable (or XSLT parameter) is defined in the XSLT Input Parameters/XQuery External Variables dialog but is not used in the XQuery

document. Neither is it an error if an XSLT parameter (or external XQuery variable) is defined in the XSLT Input Parameters/XQuery External Variables dialog but is not used in an XSLT transformation.

21.7.4 XQuery Execution



The **XSL/XQuery | XQuery Execution** command executes an XQuery document. It can be invoked when an XQuery or XML file is active. When invoked from an XML file, it opens a dialog asking for an XQuery file to associate with the XML file. You can also select a file via a global resource or a URL (click the [Browse](#) button) or a file in one of the open windows in XMLSpy (click the **Window** button).

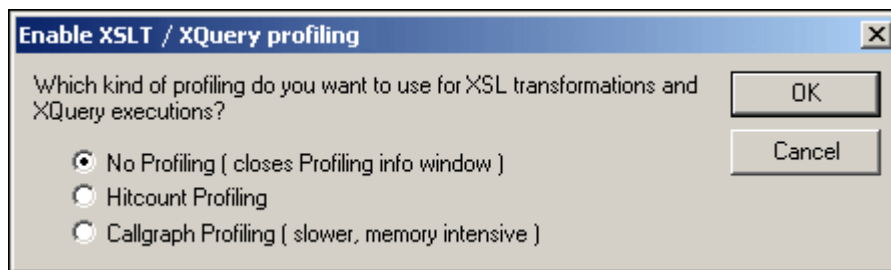
Automating XQuery executions with AltovaXML 2012

AltovaXML is a free application which contains Altova's XML Validator, XSLT 1.0, XSLT 2.0, and XQuery 1.0 engines. It can be used from the command line, via a COM interface, in Java programs, and in .NET applications to validate XML documents, transform XML documents using XSLT 1.0 and 2.0 stylesheets, and execute XQuery documents.

XQuery execution tasks can therefore be automated with the use of AltovaXML. For example, you can create a batch file that calls AltovaXML to execute a set of XQuery documents. See the [AltovaXML documentation](#) for details.

21.7.5 Enable XSLT/XQuery Profiling

The **Enable XSLT/XQuery profiling** command opens the Enable XSLT/XQuery profiling dialog. This dialog allows you to activate the [Profiler](#), which is a tool that analyzes the time it takes for instructions to execute during an XSLT transformation or XQuery execution.



21.7.6 Assign XSL



The **XSL/XQuery | Assign XSL...** command assigns an XSLT stylesheet to an XML document. Clicking the command opens a dialog to let you specify the XSLT file you want to assign. You can also select a file via a global resource or a URL (click the [Browse](#) button) or a file in one of the open windows in XMLSpy (click the **Window** button).

An `xml-stylesheet` processing instruction is inserted in the XML document:

```
<?xml-stylesheet type="text/xsl"
    href="C:\workarea\recursion\recursion.xslt"?>
```

Please note: You can make the path of the assigned file relative by clicking the **Make Path Relative To...** check box.

21.7.7 Assign XSL-FO

The **XSL/XQuery | Assign XSL:FO...** command assigns an XSLT stylesheet for transformation to FO to an XML document. The command opens a dialog to let you specify the XSL or XSLT file you want to assign and inserts the required processing instruction into your XML document:

```
<?xmlspyxslfo C:\Program Files\Altova\xmlspy\Examples\OrgChartFO.xsl?>
```

You can make the path of the assigned file relative by clicking the **Make Path Relative To...** check box. You can also select a file via a global resource or a URL (click the [Browse](#) button) or a file in one of the open windows in XMLSpy (click the **Window** button).

Please note: An XML document may have two XSLT files assigned to it: one for standard XSLT transformations, a second for an XSLT transformation to FO.

21.7.8 Assign Sample XML File



The **XSL/XQuery | Assign Sample XML File** command assigns an XML file to an XSLT document. The command inserts a processing instruction naming an XML file to be processed with this XSLT file when the XSL Transformation is executed on the XSLT file:

```
<?altova_samplexml C:\workarea\html2xml\article.xml?>
```

Please note: You can make the path of the assigned file relative by clicking the **Make Path Relative To...** check box. You can also select a file via a global resource or a URL (click the [Browse](#) button) or a file in one of the open windows in XMLSpy (click the **Window** button).

21.7.9 Go to XSL



The **XSL/XQuery | Go to XSL** command opens the associated XSLT document. If your XML document contains a stylesheet processing instruction (i.e. an XSLT assignment) such as this:

```
<?xml-stylesheet type="text/xsl" href="Company.xsl"?>
```

then the **Go to XSL** command opens the XSLT document in XMLSpy.

21.7.10 Start Debugger / Go



Alt+F11

The **XSL/XQuery | Start Debugger/Go** command starts or continues processing the XSLT/XQuery document till the end. If breakpoints have been set, then processing will pause at that point. If tracepoints have been set, output for these statements will be displayed in the Trace window when the closing node of the statement with the tracepoint has been reached. If the debugger session has not been started, then this button will start the session and stop at the first node to be processed. If the session is running, then the XSLT/XQuery document will be

processed to the end, or until the next breakpoint is encountered.

21.7.11 Stop Debugger



The **XSL/XQuery | Stop Debugger** command stops the debugger. This is not the same as stopping the debugger **session** in which the debugger is running. This is convenient if you wish to edit a document in the middle of a debugging session or to use alternative files within the same debugging session. After stopping the debugger, you must restart the debugger to start from the beginning of the XSLT/XQuery document.

21.7.12 Restart Debugger



The **XSL/XQuery | Restart Debugger** command clears the output window and restarts the debugging session with the currently selected files.

21.7.13 End Debugger Session



The **XSL/XQuery | End Debugger Session** command ends the debugging session and returns you to the normal XMLSpy view that was active before you started the debugging session. Whether the output documents that were opened for the debugging session stay open depends on a setting you make in the [XSLT/XQuery Debugger Settings](#) dialog.

21.7.14 Step Into



F11

The **XSL/XQuery | Step Into** command proceeds in single steps through all nodes and XPath expressions in the stylesheet. This command is also used to re-start the debugger after it has been stopped.

21.7.15 Step Out



Shift+F11

The **XSL/XQuery | Step Out** command steps out of the current node to the next sibling of the parent node, or to the next node at the next higher level from that of the parent node.

21.7.16 Step Over



Ctrl+F11

The **XSL/XQuery | Step Over** command steps over the current node to the next node at the same level, or to the next node at the next higher level from that of the current node. This command is also used to re-start the debugger after it has been stopped.

21.7.17 Show Current Execution Node



The **XSL/XQuery | Show Current Execution Node** command displays/selects the current execution node in the XSLT/XQuery document and the corresponding context node in the XML document. This is useful when you have clicked in other tabs which show or mark specific code in the XSLT stylesheet or XML file, and you want to return to where you were before you did this.

21.7.18 Insert/Remove Breakpoint



F9

The **XSL/XQuery | Insert/Remove Breakpoint** command inserts or removes a breakpoint at the current cursor position. Inline breakpoints can be defined for nodes in both the XSLT/XQuery and XML documents, and determine where the processing should pause. A dashed red line appears above the node when you set a breakpoint. Breakpoints cannot be defined on closing nodes, and breakpoints on attributes in XSLT documents will be ignored. This command is also available by right-clicking at the breakpoint location.

21.7.19 Insert/Remove Tracepoint



Shift+F9

The **XSL/XQuery | Insert/Remove Tracepoint** command inserts or removes a tracepoint at the current cursor position in an XSLT/XQuery document. For statements with a tracepoint, during debugging, the value of the statement is displayed in the Trace window when the closing node of that statement is reached. A dashed blue line appears above the node when you set a tracepoint. Tracepoints cannot be defined on closing nodes. This command is also available by right-clicking at the tracepoint location.

21.7.20 Enable/Disable Breakpoint



Ctrl+F9

The **XSL/XQuery | Enable/Disable Breakpoint** command (no toolbar icon exists) enables or disables already defined breakpoints. The red breakpoint highlight turns to gray when the breakpoint is disabled. The debugger does not stop at disabled breakpoints. To disable/enable a breakpoint, place the cursor in that node name and click the **Enable/Disable Breakpoint** command. This command is also available by right-clicking at the location where you want to enable/disable the breakpoint.

21.7.21 Enable/Disable Tracepoint



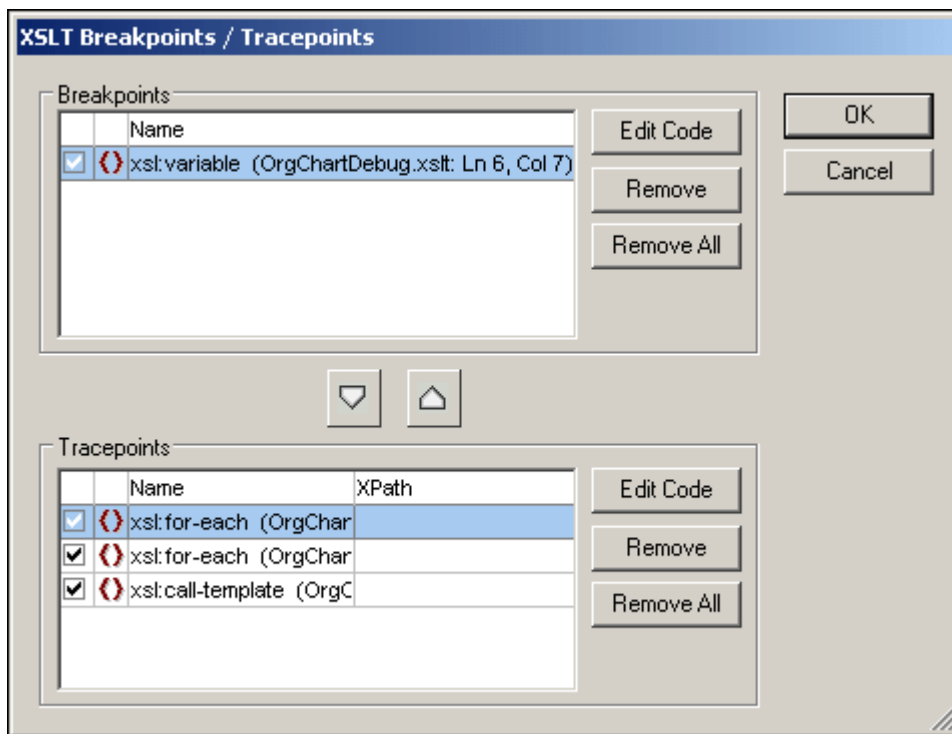
Ctrl+Shift+F9

The **XSL/XQuery | Enable/Disable Tracepoint** command (no toolbar icon exists) enables or disables already defined tracepoints. The blue tracepoint highlight turns to gray when the tracepoint is disabled. No output is displayed for statements with disabled tracepoints. To disable/enable a tracepoint, place the cursor in that node name and click the **Enable/Disable Tracepoint** command. This command is also available by right-clicking at the location where you want to enable/disable the tracepoint.

21.7.22 Breakpoints/Tracepoints



The **XSL/XQuery | Breakpoints/Tracepoints...** command opens the XSLT Breakpoints / Tracepoints dialog, which displays a list of all currently defined breakpoints and tracepoints (including disabled breakpoints and tracepoints) in all files in the current debugging session.



The check boxes indicate whether a breakpoint or tracepoint is enabled (checked) or disabled. You can remove the highlighted breakpoint or tracepoint by clicking the corresponding **Remove** button, and remove all breakpoints by clicking the corresponding **Remove All** button. The **Edit Code** button takes you directly to that breakpoint/tracepoint in the file.

Use the down arrow  to move the highlighted breakpoint to the **Tracepoints** pane and the up arrow  to move the highlighted tracepoint to the **Breakpoints** pane.

In the **XPath** column in the Tracepoints pane, you can set an XPath for each tracepoint.

21.7.23 Debug Windows

Placing the cursor over the **XSL/XQuery | Debug Windows** command pops out a submenu with a list of the various Information Windows of the XSLT/XQuery Debugger. Selecting an Information Window from this list shows/hides that Information Window in the XSLT/XQuery Debugger interface. This command can be used to effect only when a debugging session is in progress.

21.7.24 XSLT/XQuery Settings



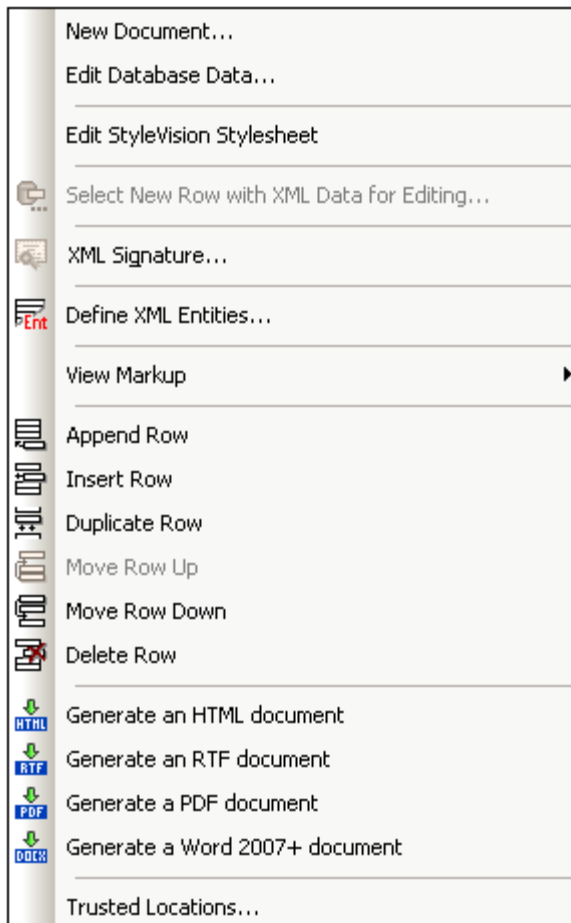
The **XSL/XQuery | XSLT/XQuery Settings** command opens the [XSLT/XQuery Settings dialog](#), which enables you to set user options for the Debugger. See the [XSLT/XQuery Debugger](#) section for details.

21.8 Authentic Menu

Authentic View enables you to edit XML documents **based on StyleVision Power Stylesheets (.sps files) created in Altova's StyleVision product!** These stylesheets contain information that enables an XML file to be displayed graphically in Authentic View. In addition to containing display information, StyleVision Power Stylesheets also allow you to write data to the XML file. This data is dynamically processed using all the capability available to XSLT stylesheets and instantly produces the output in Authentic View.

Additionally, StyleVision Power Stylesheets can be created to display an editable XML view of a database. The StyleVision Power Stylesheet contains information for connecting to the database, displaying the data from the database in Authentic View, and writing back to the database.

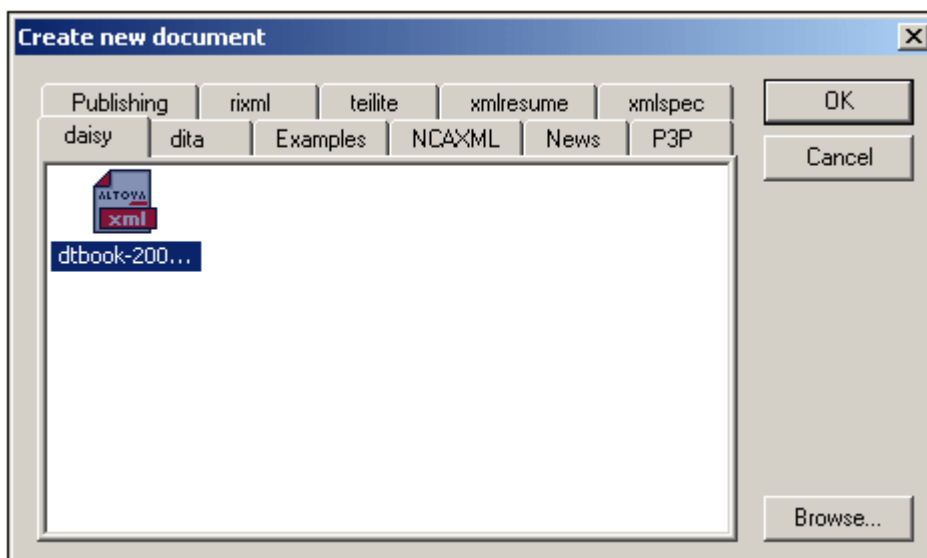
The **Authentic** menu contains commands relevant to editing XML documents in Authentic View. For a tutorial on Authentic View, see the [Tutorials](#) section.



21.8.1 New Document

The **Authentic | New Document...** command enables you to open a new XML document template in Authentic View. The XML document template is based on a StyleVision Power Stylesheet (.sps file), and is opened by selecting the StyleVision Power Stylesheet.

Clicking the **New Document...** command opens the Create New Document dialog.



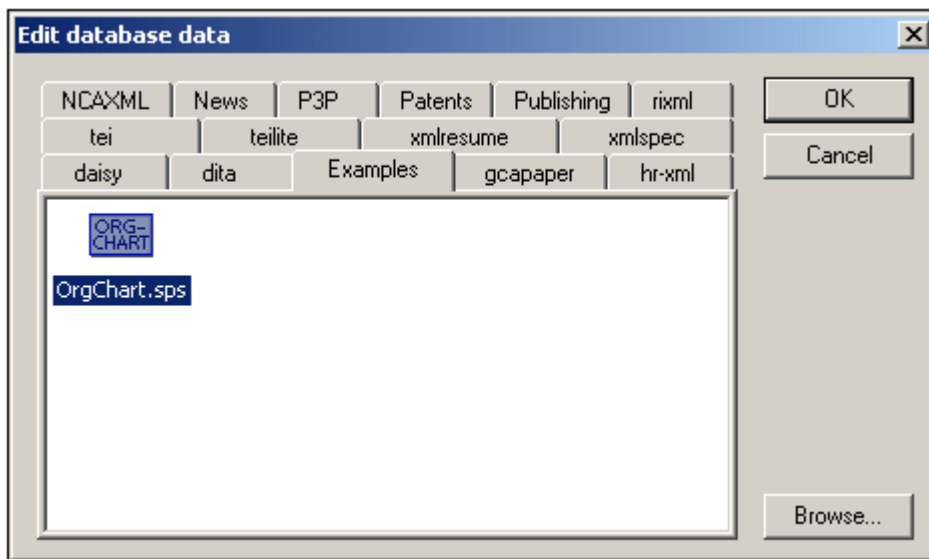
Browse for the required SPS file, and select it. This opens an XML document template in Authentic View.

Note: StyleVision Power Stylesheets are created using Altova StyleVision. The StyleVision Power Stylesheet has a Template XML File assigned to it. The data in this XML file provides the starting data of the new document template that is opened in Authentic View.

21.8.2 Edit Database Data

The **Authentic | Edit Database Data...** command enables you to open an editable view of a database (DB) in Authentic View. All the information about connecting to the DB and how to display the DB and accept changes to it in Authentic View is contained in a StyleVision Power Stylesheet. It is such a DB-based StyleVision Power Stylesheet that you open with the **Edit Database Data...** command. This sets up a connection to the DB and displays the DB data (through an XML lens) in Authentic View.

Clicking the **Edit Database Data...** command opens the Edit Database Data dialog.



Browse for the required SPS file, and select it. This connects to the DB and opens an editable view of the DB in Authentic View. The design of the DB view displayed in Authentic View is contained in the StyleVision Power Stylesheet.

Please note: If, with the **Edit Database Data...** command, you attempt to open a StyleVision Power Stylesheet that is not based on a DB or to open a DB-based StyleVision Power Stylesheet that was created in a version of StyleVision prior to the StyleVision 2005 release, you will receive an error.

Please note: StyleVision Power Stylesheets are created using Altova StyleVision.

21.8.3 Assign/Edit a StyleVision Stylesheet

Assign a StyleVision Stylesheet

The **Assign a StyleVision Stylesheet** command assigns a StyleVision Power Stylesheet (SPS) to an **XML document** to enable the viewing and editing of that XML document in Authentic View. The StyleVision Power Stylesheet that is to be assigned to the XML file must be based on the same schema as that on which the XML file is based.

To assign a StyleVision Power Stylesheet to an XML file:

1. Make the XML file the active file and select the **Authentic | Assign a StyleVision Stylesheet...** command.
2. The command opens a dialog box in which you specify the StyleVision Power Stylesheet file you wish to assign to the XML.
3. Click **OK** to insert the required SPS statement into your XML document. Note that you can make the path to the assigned file relative by clicking the **Make path relative to ...** check box. You can also select a file via a global resource or a URL (click the [Browse](#) button) or a file in one of the open windows in XMLSpy (click the **Window** button).

```
<?xml version="1.0" encoding="UTF-8"?>
<?altova_sps HTML-Orgchart.sps?>
```

In the example above, the StyleVision Power Stylesheet is called `HTML_Orgchart.sps`, and it is located in the same directory as the XML file.

Please note: Previous versions of Altova products used a processing instruction with a target or name of `xmlspysps`, so a processing instruction would look something like `<?xmlspysps`

HTML-Orgchart.sps?>. These older processing instructions are still valid with Authentic View in current versions of Altova products.

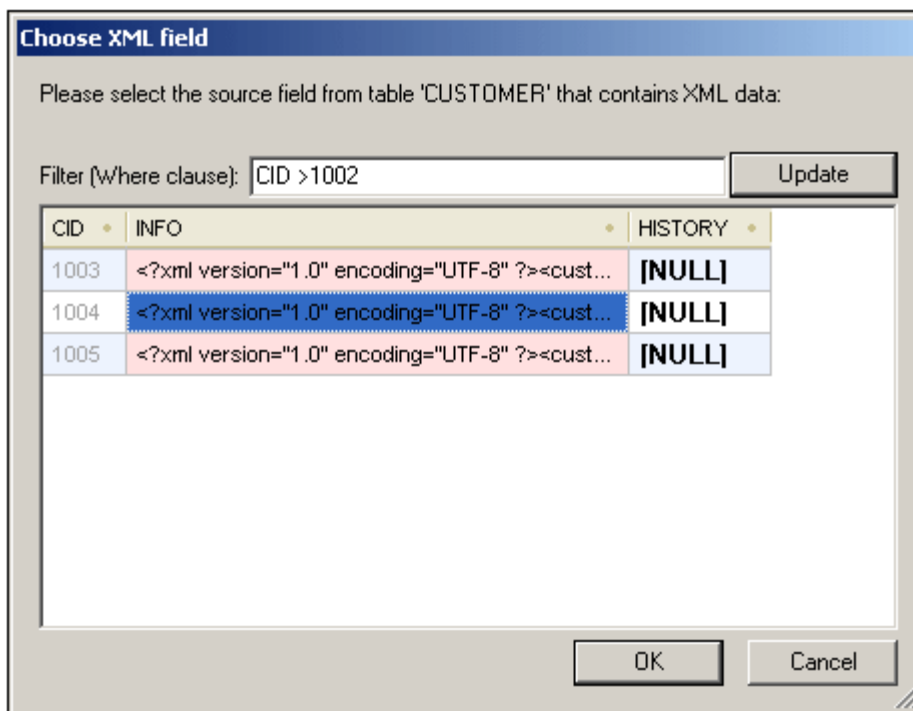
Edit StyleVision Stylesheet

The **Authentic | Edit StyleVision Stylesheet** command starts StyleVision and allows you to edit the StyleVision Power Stylesheet immediately in StyleVision.

21.8.4 Select New Row with XML Data for Editing

The **Select New Row with XML Data for Editing** command enables you to select a new row from the relevant table in an XML DB, such as IBM DB2. This row appears in Authentic View, can be edited there, and then saved back to the DB.

When an XML DB is used as the XML data source, the XML data that is displayed in Authentic View is the XML document contained in one of the cells of the XML data column. The **Select New Row with XML Data for Editing** command enables you to select an XML document from another cell (or row) of that XML column. Selecting the **Select New Row...** command pops up the Choose XML Field dialog (*screenshot below*), which displays the table containing the XML column.



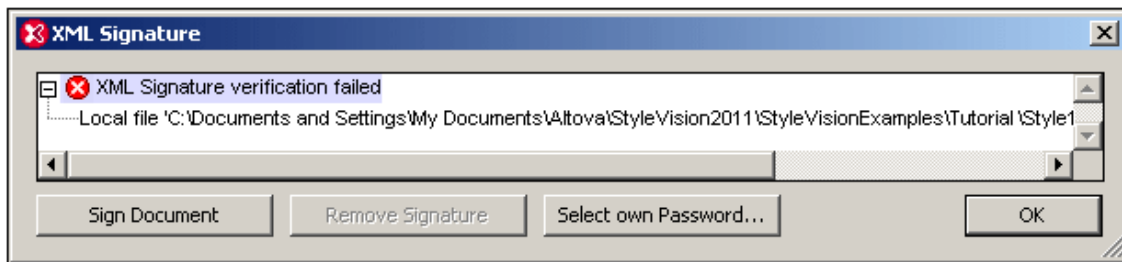
You can enter a filter for this table. The filter should be an SQL `WHERE` clause (just the condition, without the `WHERE` keyword, for example: `CID>1002`). Click **Update** to refresh the dialog. In the screenshot above, you can see the result of a filtered view. Next, select the cell containing the required XML document and click **OK**. The XML document in the selected cell (row) is loaded into Authentic View.

21.8.5 XML Signature

The **XML Signature** command is available in Authentic View when the associated SPS has XML Signatures enabled. The **XML Signature** command is also available as the XML Signature toolbar icon  in the Authentic toolbar.

Verification and own certificate/password

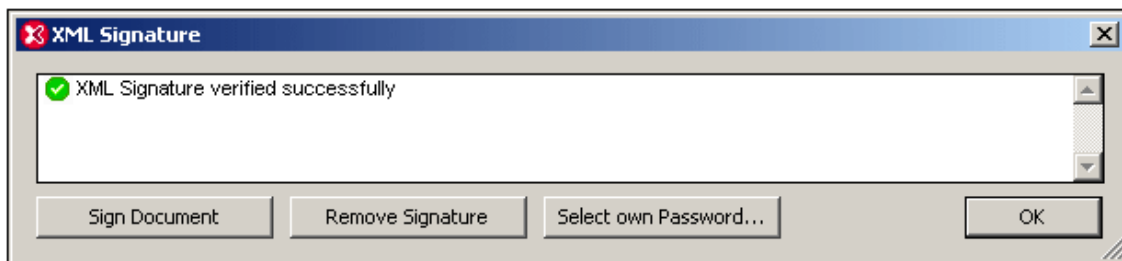
Clicking the **XML Signature** command starts the signature verification process. If no signature is present in the document, a message to that effect is displayed in the XML Signature dialog (see *screenshot below*), and the dialog will have a button that enables the Authentic View user to sign the document.



If the **Select Own Certificate** or **Select Own Password** button is present in this dialog, it means that the Authentic View has been given the option of selecting an own certificate/password. (Whether a certificate or password is to be chosen has been decided by the SPS designer at the time the signature was configured. The signature will be either certificate-based or password-based.) Clicking either of these buttons, if present in the dialog, enables the Authentic View user to browse for a certificate or to enter a password. The Authentic View user's selection is stored in memory and is valid for the current session only. If, after selecting a certificate or password, the document or application is closed, the certificate/password setting reverts to the setting originally saved with the SPS.

Verification and authentication information

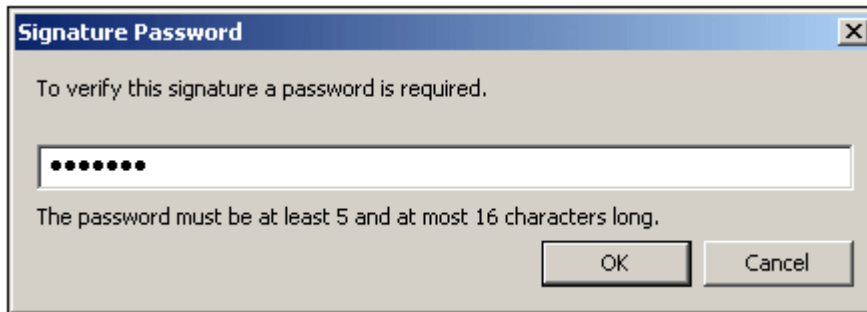
If the verification process is run on a signed document, two general situations are possible. First: If the authentication information is available (in the signature or the SPS), then the verification process is executed directly and the result is displayed (*screenshot below*).



Authentication information is either the signing certificate's key information or the signing password. The SPS designer will have specified whether the certificate's key information is saved in the signature when the XML document is signed, or, in the case of a password-based signature, whether the password is saved in the SPS. In either of these cases, the authentication is available. Consequently the verification process will be run directly, without requiring any input from the Authentic View user.

The second possible general situation occurs when authentication information is not available in

the signature (certificate's key information) or SPS file (password). In this situation, the Authentic View user will be asked to supply the authentication information: a password (see *screenshot below*) or the location of a certificate.



21.8.6 Define XML Entities

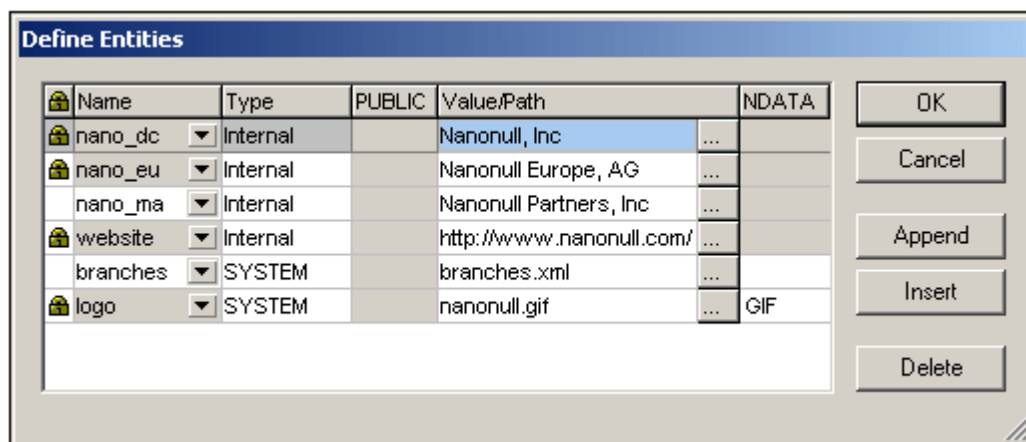
You can define entities for use in Authentic View, whether your document is based on a DTD or an XML Schema. Once defined, these entities are displayed in the Entities Entry Helper and in the **Insert Entity** submenu of the context menu. When you double-click on an entity in the Entities Entry Helper, that entity is inserted at the cursor insertion point.

An entity is useful if you will be using a text string, XML fragment, or some other external resource in multiple locations in your document. You define the entity, which is basically a short name that stands in for the required data, in the Define Entities dialog. After defining an entity you can use it at multiple locations in your document. This helps you save time and greatly enhances maintenance.

There are two broad types of entities you can use in your document: a **parsed entity**, which is XML data (either a text string or a fragment of an XML document), or an **unparsed entity**, which is non-XML data such as a binary file (usually a graphic, sound, or multimedia object). Each entity has a name and a value. In the case of parsed entities the entity is a placeholder for the XML data. The value of the entity is either the XML data itself or a URI that points to a .xml file that contains the XML data. In the case of unparsed entities, the value of the entity is a URI that points to the non-XML data file.

To define an entity:

1. Click **Authentic | Define XML Entities....** This opens the Define Entities dialog.



2. Enter the name of your entity in the **Name** field. This is the name that will appear in the Entities Entry Helper.

3. Enter the type of entity from the drop-down list in the **Type** field. Three types are possible. An **Internal** entity is one for which the text to be used is stored in the XML document itself. Selecting **PUBLIC** or **SYSTEM** specifies that the resource is located outside the XML file, and will be located with the use of a public identifier or a system identifier, respectively. A system identifier is a URI that gives the location of the resource. A public identifier is a location-independent identifier, which enables some processors to identify the resource. If you specify both a public and system identifier, the public identifier resolves to the system identifier, and the system identifier is used.
4. If you have selected PUBLIC as the Type, enter the public identifier of your resource in the PUBLIC field. If you have selected Internal or SYSTEM as your Type, the PUBLIC field is disabled.
5. In the **Value/Path** field, you can enter any one of the following:
 - If the entity type is Internal, enter the text string you want as the value of your entity. Do not enter quotes to delimit the entry. Any quotes that you enter will be treated as part of the text string.
 - If the entity type is SYSTEM, enter the URI of the resource or select a resource on your local network by using the **Browse** button. If the resource contains parsed data, it must be an XML file (i.e. it must have a .xml extension). Alternatively, the resource can be a binary file, such as a GIF file.
 - If the entity type is PUBLIC, you must additionally enter a system identifier in this field.
6. The NDATA entry tells the processor that this entity is not to be parsed but to be sent to the appropriate processor. The NDATA field should therefore be used with unparsed entities only.

Dialog features

You can append, insert, and delete entities by clicking the appropriate buttons. You can also sort entities on the alphabetical value of any column by clicking the column header; clicking once sorts in ascending order, twice in descending order. You can also resize the dialog box and the width of columns.

Once an entity is used in the XML document, it is locked and cannot be edited in the Define Entities dialog. Locked entities are indicated by a lock symbol in the first column. Locking an entity ensures that the XML document valid with respect to entities. (The document would be invalid if an entity is referenced but not defined.)

Duplicate entities are flagged.

Limitations

- An entity contained within another entity is not resolved, either in the dialog, Authentic View, or XSLT output, and the ampersand character of such an entity is displayed in its escaped form, i.e. & amp; .
- External entities are not resolved in Authentic View, except in the case where an entity is an image file and it is entered as the value of an attribute which has been defined in the schema as being of type `ENTITY` or `ENTITIES`. Such entities are resolved when the document is processed with an XSLT generated from the SPS.

21.8.7 View Markup

The **View Markup** command has a submenu with options to control markup in the Authentic XML document. These options are described below.



The **Hide Markup** command hides markup symbols in Authentic View.



The **Show Small Markup** command shows small markup symbols in Authentic View.



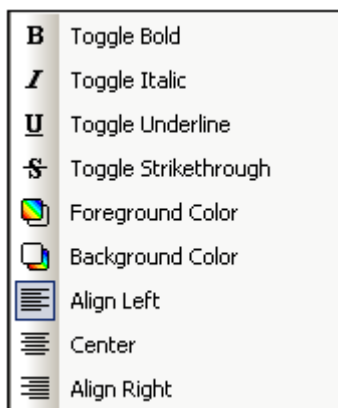
The **Show Large Markup** command shows large markup symbols in Authentic View.



The **Show Mixed Markup** command shows mixed markup symbols in Authentic View. The person who designs the StyleVision Power Stylesheet can specify either large markup, small markup, or no markup for individual elements/attributes in the document. The Authentic View user sees this customized markup in mixed markup viewing mode.

21.8.8 RichEdit

Mousing over the RichEdit command pops out a submenu containing the RichEdit markup commands (*screenshot below*). The menu commands in this submenu are enabled only in Authentic View and when the cursor is placed inside an element that has been created as a RichEdit component in the SPS design.



The text-styling properties of the RichEdit menu will be applied to the selected text when a RichEdit command is clicked. The Authentic View user can specify the font, font-weight, font-style, font-decoration, font-size, color, background color and alignment of the selected text.

21.8.9 Append/Insert/Duplicate/Delete Row



The **Append Row** command appends a row to the current table in Authentic View.



The **Insert Row** command inserts a row into the current table in Authentic View.



The **Duplicate Row** command duplicates the current table row in Authentic View.



The **Delete Row** command deletes the current table row in Authentic View.

21.8.10 Move Row Up/Down



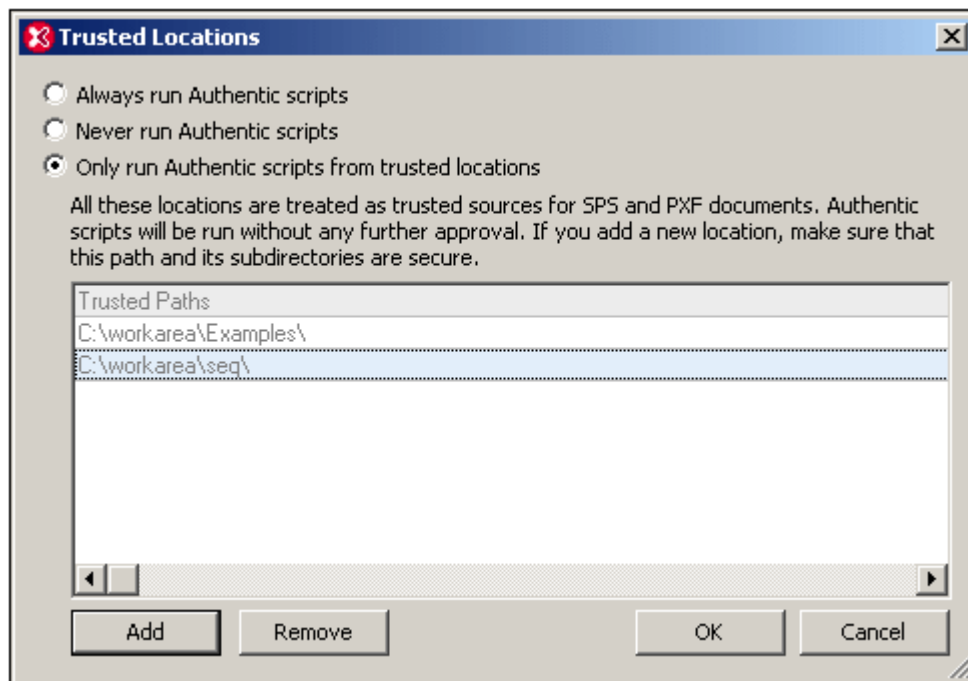
The **Move Row Up** command moves the current table row up by one row in Authentic View.



The **Move Row Down** command moves the current table row down by one row in Authentic View.

21.8.11 Trusted Locations

The Trusted Locations command pops up the Trusted Locations dialog (*screenshot below*), in which you can specify the security settings for scripts in an SPS. When an XML file based on a script-containing SPS is switched to Authentic View, the script will be allowed to run or not depending on the settings you make in this dialog.



The three available options are:

- Authentic scripts are always run when a file is opened in Authentic View.
- Authentic scripts are never run when a file is opened in Authentic View.
- Only Authentic scripts in trusted locations are run. The list of trusted (folder) locations is shown in the bottom pane. Use the **Add** button to browse for a folder and add it to the list. To remove an entry from the list, select an entry in the Trusted Locations list and click **Remove**.

21.9 DB Menu

The **DB** menu contains the following menu items:

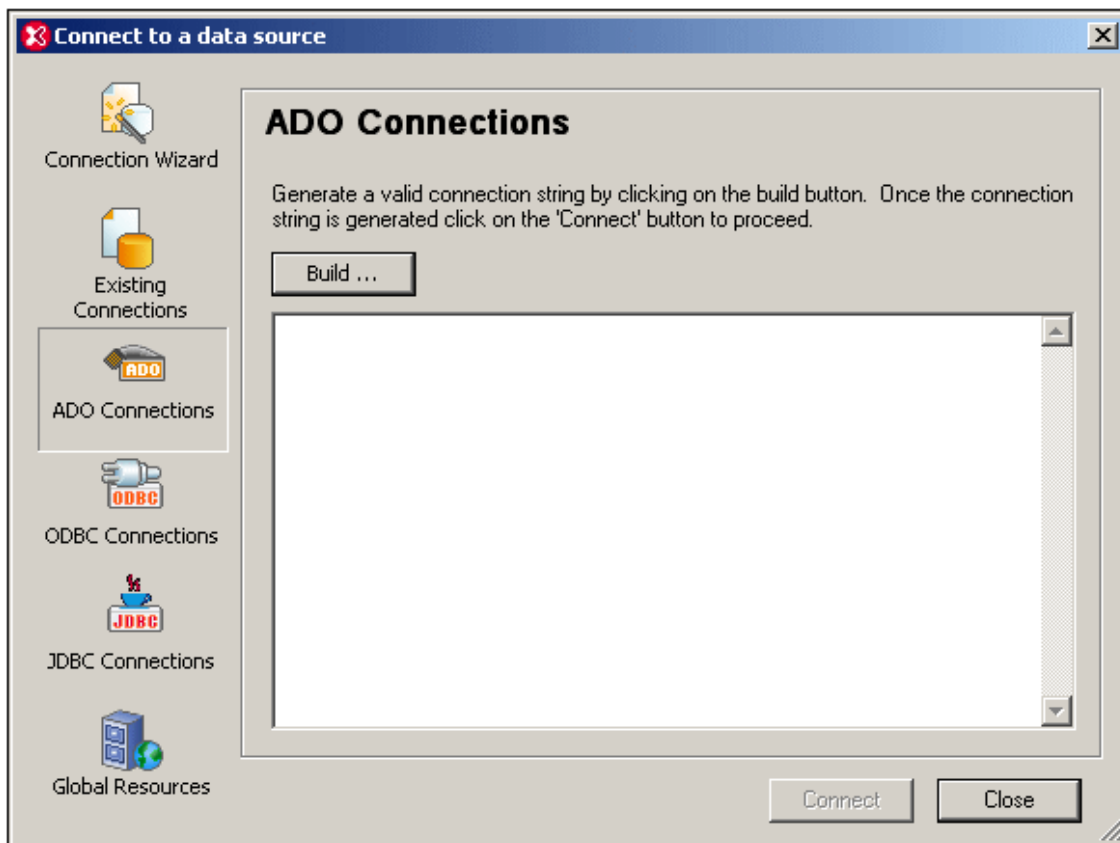
- [Query Database](#), which enables you to query a variety of databases.
- [IBM DB2](#), which contains commands that provide support for DB2-specific functionality.
- [SQL Server](#), which contains commands for managing SQL Server databases.
- [Oracle databases](#), which contains command for working with Oracle databases.

Since all the operations described in this section require a connection to a database, this section also includes a sub-section called [Connecting to a Data Source](#).

Note: If you are using the 64-bit version of XMLSpy, ensure that you have access to the 64-bit database drivers needed for the specific database you are connecting to.

21.9.1 Connecting to a Data Source

A number of commands and icons in the **DB** and **Convert** menus require a connection to a database. XMLSpy enables you to connect to a variety of databases (*see list below*). It uses the Connect to Data Source dialog (for commands in the **Convert** menu, *screenshot below*) or Quick Connect dialog (for the **DB | Query Database** command) to set up and make the connection. Both dialogs are essentially the same; they guide you through the same connection steps. In this section, we describe how the mechanism behind these two dialogs works. In descriptions of the **DB** or **Convert** commands that require a database connection, refer to this section for information on making the connection.



The options for connecting to the database are:

- Using a [Connection Wizard](#) that guides you through the connection process
- Using an [existing connection](#)
- Using an [ADO Connection](#)
- Using an [ODBC Connection](#)
- Using a [JDBC Connections](#)
- Using a [global resource](#)

Each option is described in the subsections of this section. The descriptions in this section explain only the connection mechanism. The dialogs that appear subsequent to the connection process are dependent on the specific command being executed, and these dialogs are therefore described in the sections relating to that specific command. For example, if you are looking for a description of the command [Convert | Import Database Data](#), the description for the connection process is in this section, but the selection of DB tables is described in the section [Convert | Import Database Data](#).

Supported databases

The following databases are supported. After connecting to the database, the available root objects for each of the supported databases is as follows:

Database (natively supported)	Root Object
MS SQL Server 2000, 2005, and 2008	database
MS SQL Server 2005 and 2008	schema
Oracle 9i, 10g, and 11g	schema
MS Access 2003 and 2007	database
MySQL 4.x and 5.x	database
IBM DB2 8.x and 9	schema
Sybase 12	database
IBM DB2 for i 5.4 and i 6.1	schema
PostgreSQL 8.0, 8.1, 8.2, 8.3	database

Note: XMLSpy fully supports the databases listed above. While Altova endeavors to support other ODBC/ADO databases, successful connection and data processing have only been tested with the listed databases.

Connection Wizard

In the Connect to Data Source dialog, when the **Connection Wizard** button is selected, the Connection Wizard (*screenshot below*) pops up. The Connection Wizard helps you to connect to the most commonly used types of databases. In this section, we go through the connection to a Microsoft Access database and an IBM DB2 database.

MS Access

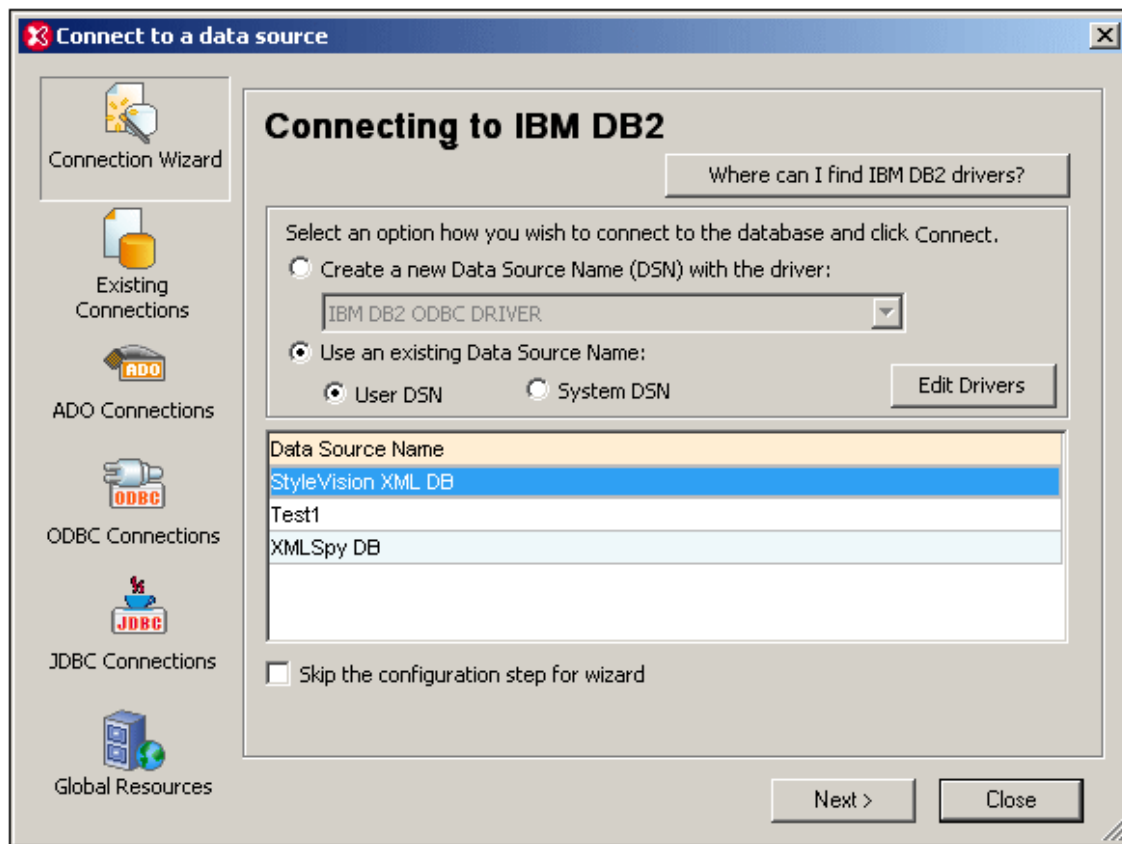
Carry out the following steps to connect to an MS Access database. Select Microsoft Access (ADO) (*screenshot below*), and click **Next**.



In the Connect to MS Access dialog that pops up, browse for the MS Access database, and click **Next**. The connection to the data source is established. For information about subsequent dialogs, refer to the description of the command being executed.

IBM DB2

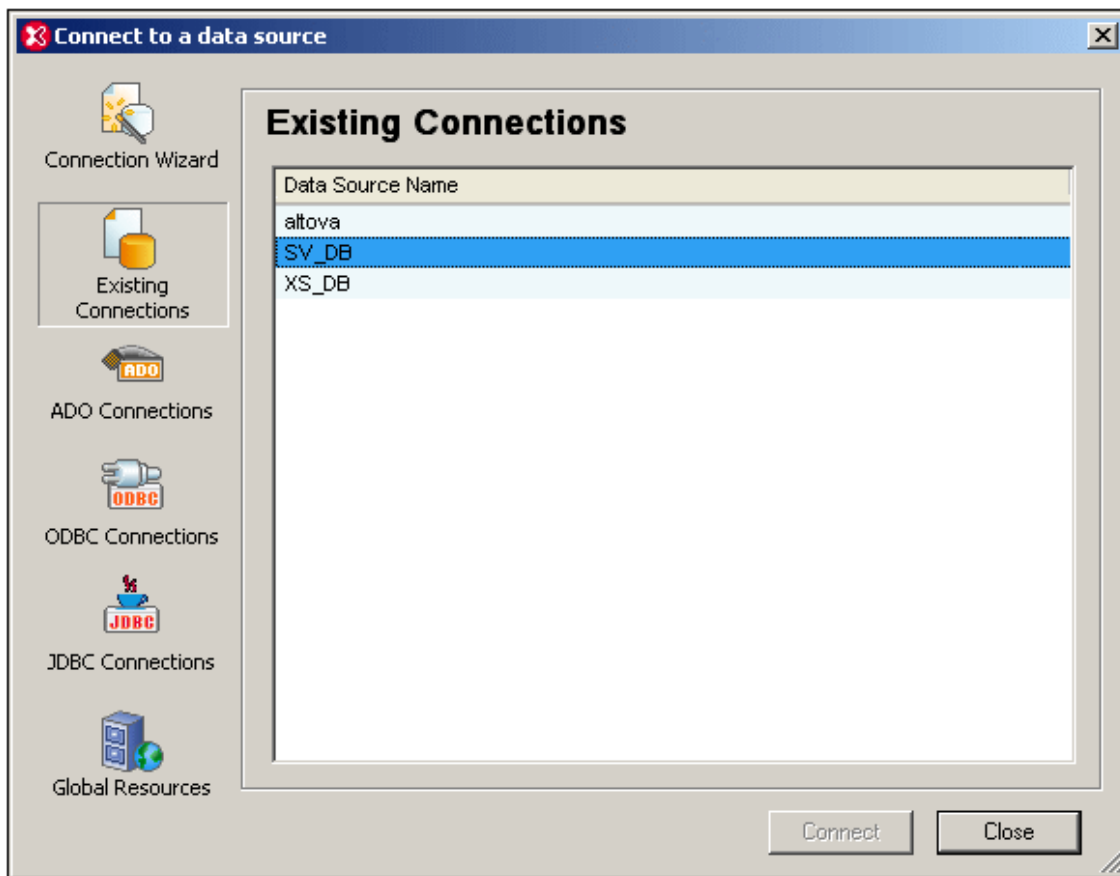
In the first screen of the Connection Wizard (*see first screenshot in this section*), select IBM DB2 and click **Next**. The Connection dialog (*screenshot below*) pops up. The wizard will then guide you in configuring the connection to the IBM DB2 database and making the connection. These steps consist essentially of selecting the appropriate driver to connect to the DB, then locating the DB, and entering user information that enables you to connect.



In the Configuration dialog above, select an existing data source, or create a new data source with an appropriate driver (additional drivers can be added to the list of available drivers by clicking the **Edit Drivers** button and selecting the required drivers). In the following screen you will be prompted for user information (ID and password). Clicking **OK** will establish the connection with the database. For information about subsequent dialogs, refer to the description of the command being executed.

Existing Connections

In the Connect to Data Source dialog, when the **Existing Connections** button is selected, the Existing Connections pane opens (*screenshot below*), showing all the connections that are currently established.

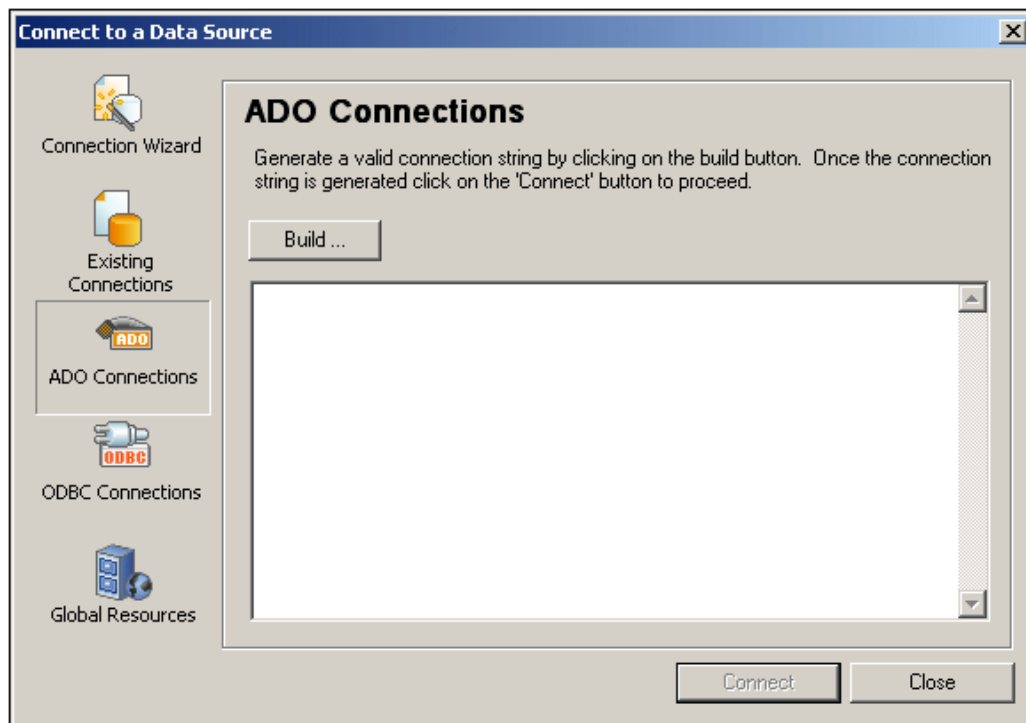


Select the required database and click **Connect**. The connection to the database is established. For information about subsequent dialogs, refer to the description of the command being executed.

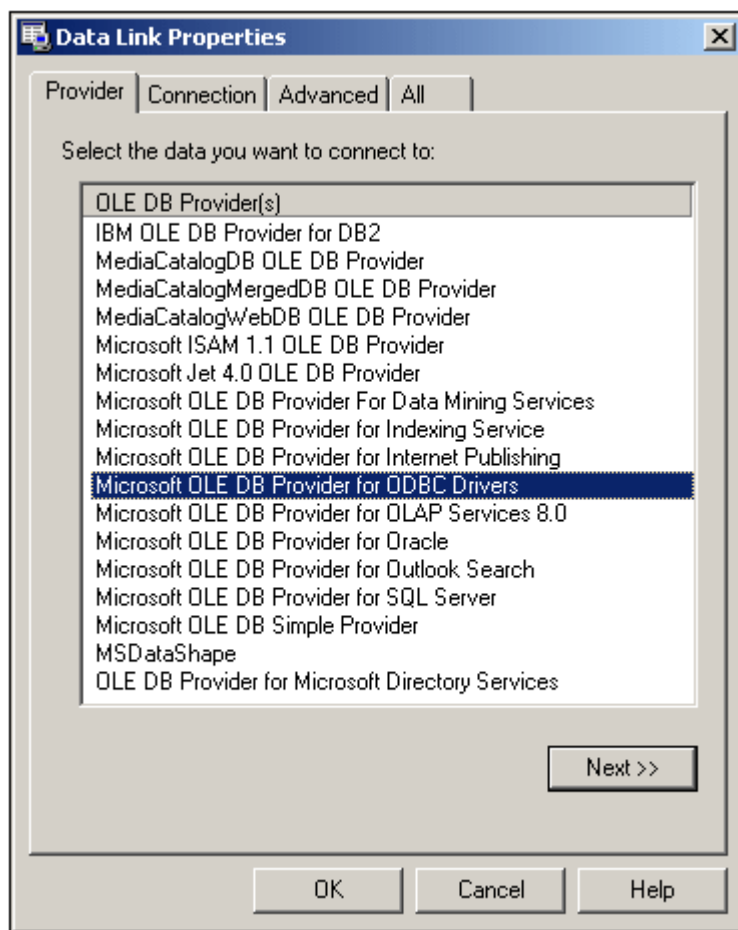
ADO Connections

The ADO Connections option enables you to connect to a DB via ADO. In this section, the connection via ADO is described using an IBM DB2 database as the target DB. Where options are different if another type of database is the target DB, these differences are noted.

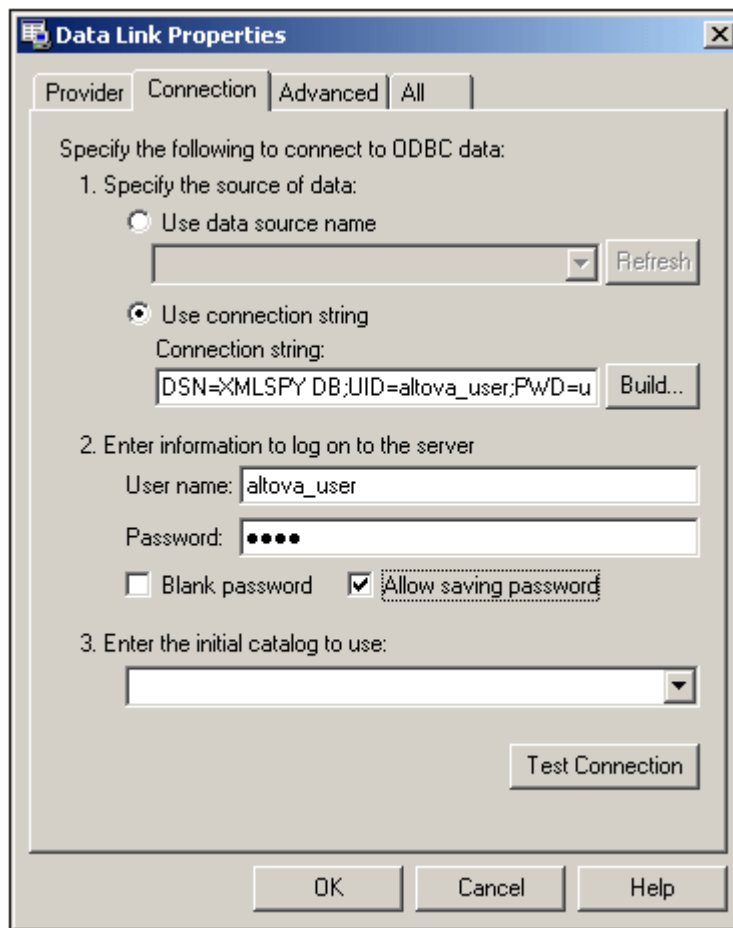
1. In the Connect to Data Source dialog, select ADO Connections. The ADO Connections box appears (*screenshot below*).



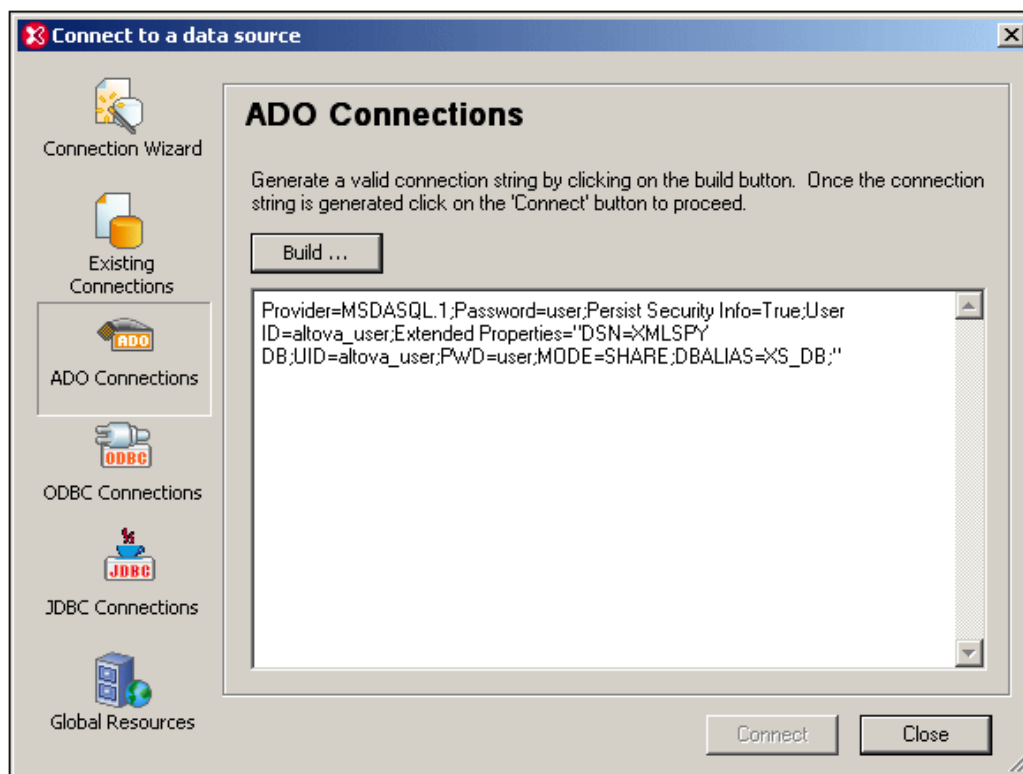
2. Click the **Build** button. The Data Link Properties dialog (*screenshot below*) opens.



3. Select Microsoft OLE DB Provider for ODBC Drivers from the drop-down list and click **Next**. The Connection tab is activated. (For Microsoft SQL DBs, we recommend *Microsoft OLE DB Provider for SQL Server*; for Oracle, MySQL, Sybase, and IBM DB2 DBs, *Microsoft OLE DB Provider for ODBC Drivers*.)



4. Build a connection string via the **Build** button (the dialog which pops up prompts you for a data source and user information). In the case of some databases (such as Microsoft SQL Server), you do not build a connection string but instead select the server and database from combo box listings.
5. If login information is required, enter a user name and password, and check the Allow Saving Password check box.
6. Click **OK**. The Data Link Properties dialog closes. The connection string appears in the ADO Connections box (*screenshot below*).

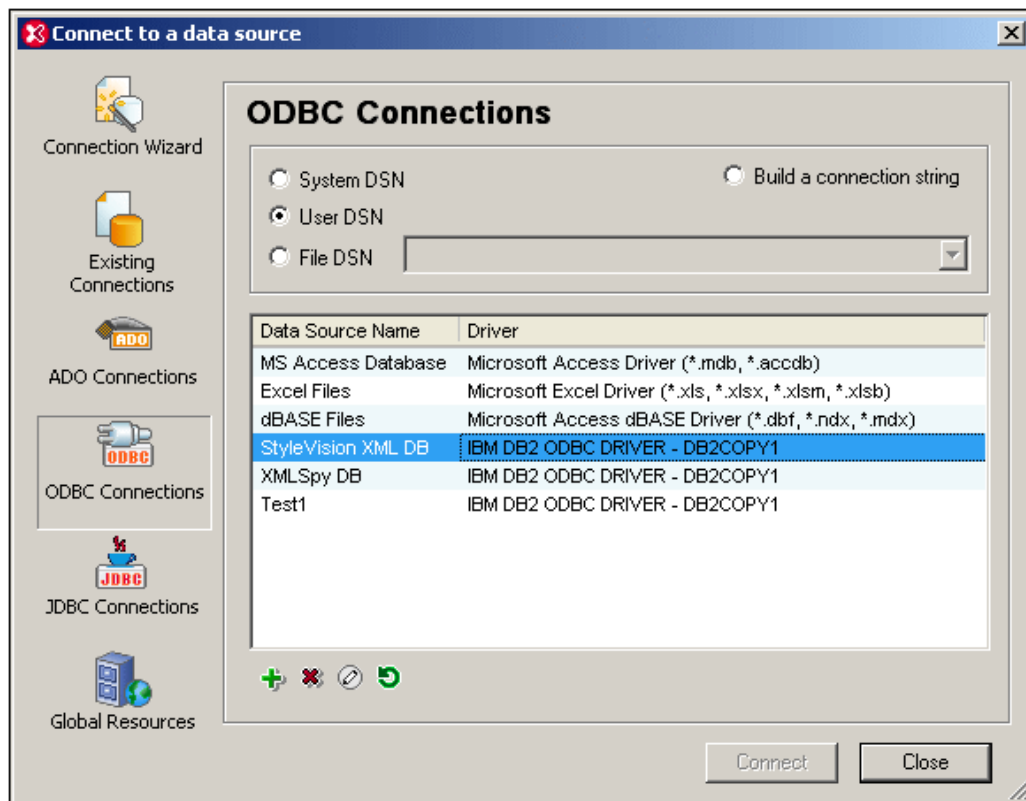


7. Click the **Connect** button. The connection to the data source is established. For information about subsequent dialogs, refer to the description of the command being executed.

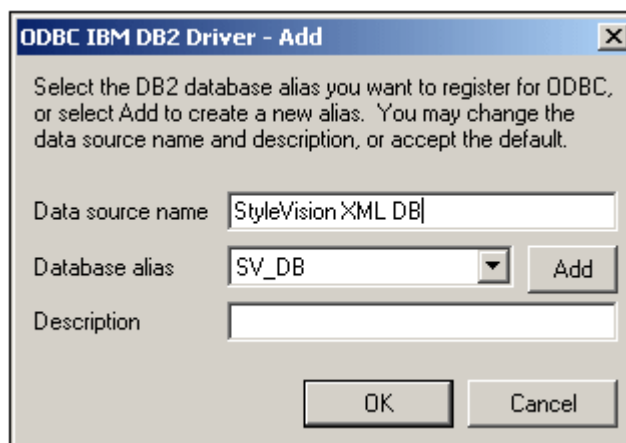
ODBC Connections

This section describes how to connect to a DB via ODBC. The steps listed below describe a connection to an IBM DB2 database, but apply also to other types of databases that can be connected via ODBC. To connect using an ODBC connection, do the following:

1. In the Connect to Data Source dialog, select **ODBC Connections**.



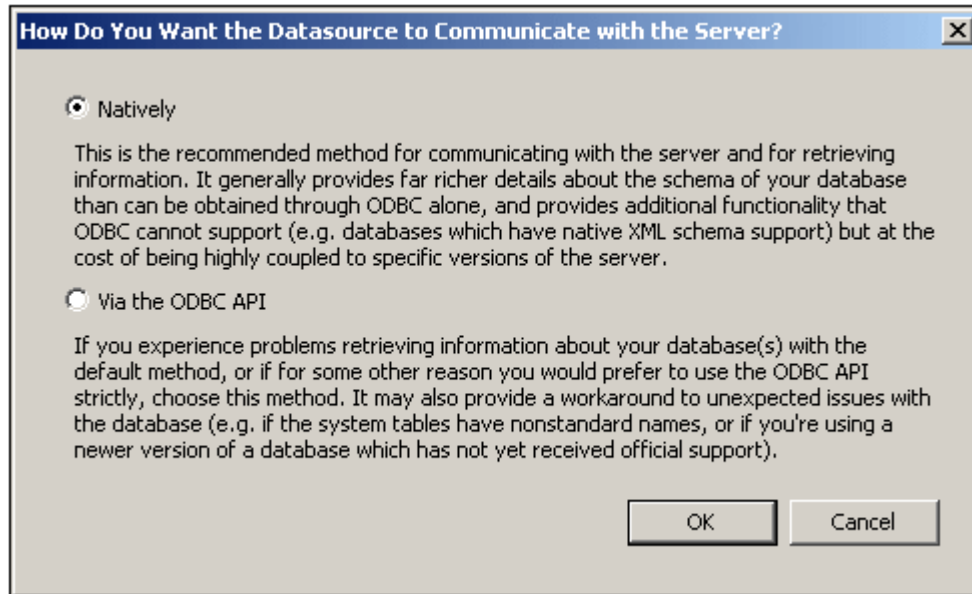
2. Select one of the options for specifying a Data Source Name (DSN). If you select System DSN or User DSN, the available DSNs are displayed in the Data Source pane. If you select File DSN, you are prompted to browse for the DSN file. Alternatively, you can build a connection string to the DB by selecting the Build a Connection String option.
3. If you wish to add a DSN to those in the Data Source pane, click the Create a New DSN icon .
4. In the Create an ODBC DSN dialog that appears, select the required driver, then click the **User DSN** or **System DSN** button.
5. In the dialog that appears (*screenshot below*), select the DB alias and give it a DSN.



6. Click **OK** to finish. The DB is added as a Data Source to the list in the Data Source pane.
7. After you have selected the DataSource (via the System DSN or User DSN option), or

selected a DSN File (File DSN option), or built a connection string (Connection String option), click **Connect**.

8. When you are prompted for your user ID and password, enter these and then click **OK**.
9. You will be asked to choose between connecting to the database either natively or via the ODBC API (see *screenshot below*).



Select the *Natively* option, unless there are difficulties with the connection or if you prefer to use the ODBC API. Then click **OK**. The connection is established. For information about subsequent dialogs, refer to the description of the command being executed.

Note: The listing in the data source pane (when the System DSN or User DSN option is selected) can be edited using the *Edit Selected DSN*, *Remove Selected DSN*, and *Refresh Listed DSNs* icons at the bottom of the ODBC Connections screen.



Edit selected DSN.



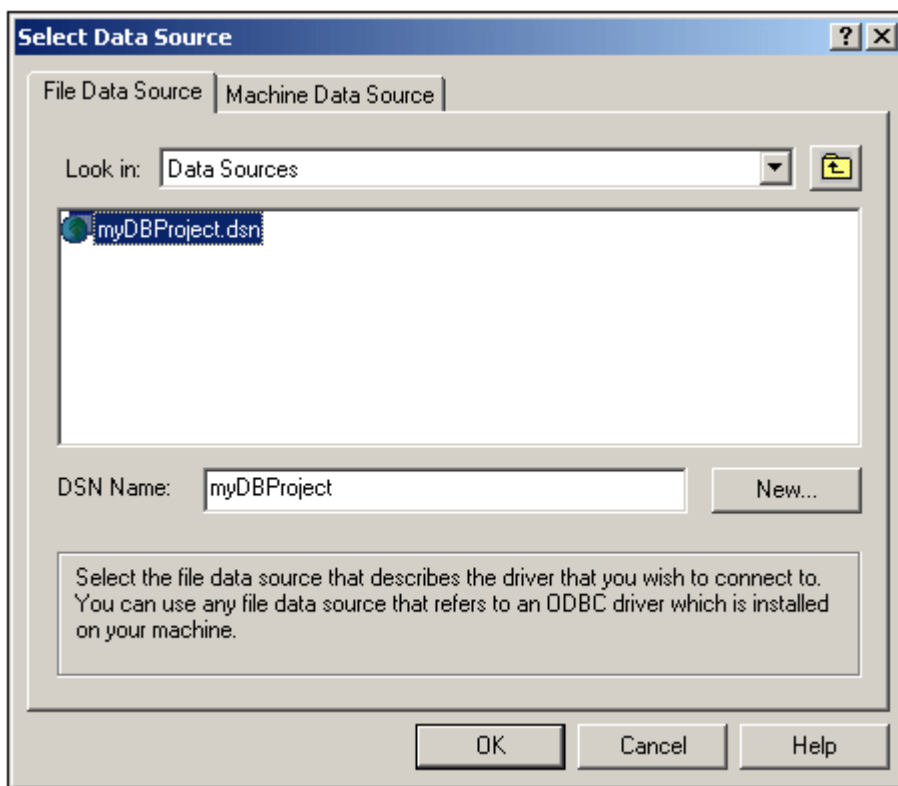
Remove selected DSN.



Refresh listed DSNs.

Building a connection string

In the ODBC Connections screen, select the Build a Connection String radio button and then click the **Build** button. The Select Data Source dialog (*screenshot below*) pops up. You can either select a File DSN (in the File Data Source tab), or select a data source that is available on your machine (listed in the Machine Data Source tab).



Clicking **OK** pops up a dialog that prompts for user information, including the User ID and password. When you click **OK** in this dialog, the connection string is entered in the ODBC Connections screen.

JDBC Connections

This section describes how to connect to a DB via JDBC. The steps listed below describe a connection to an IBM DB2 database, but they apply also to other types of databases that can be connected to via JDBC.

Pre-connection steps

Before connecting to the DB via JDBC, you must do the following:

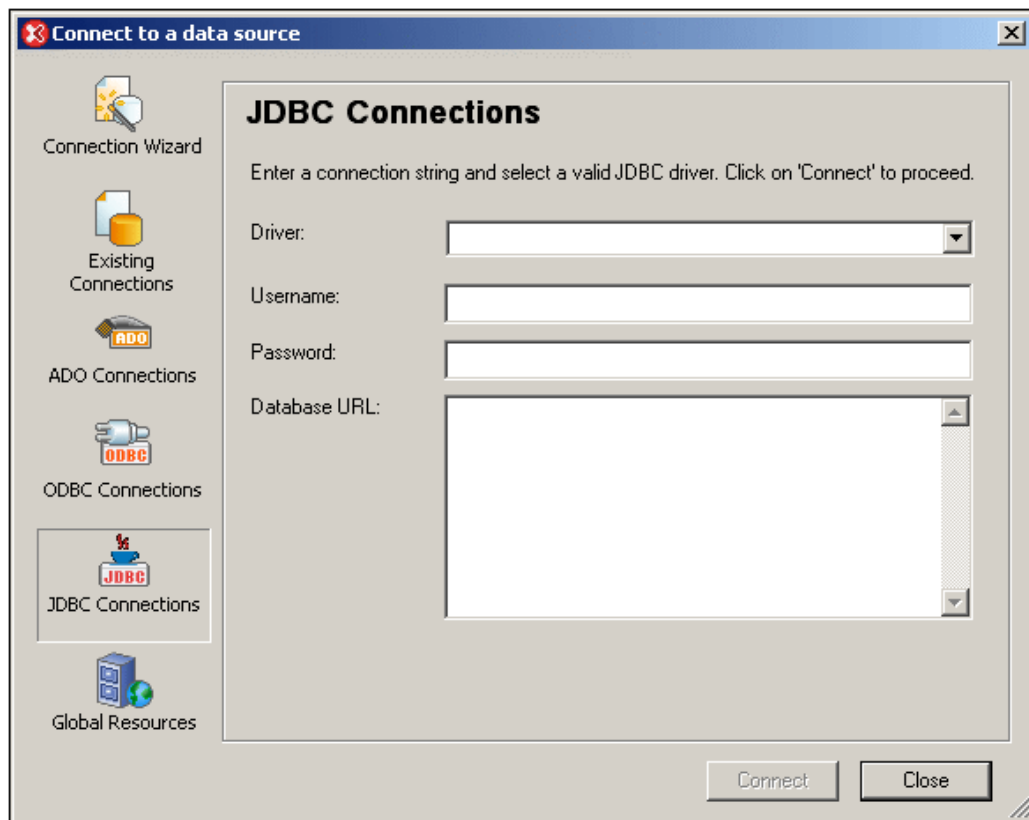
1. Ensure that the Java Runtime Environment (JRE) is installed. If it isn't installed, then install it. Use a 32-bit JRE for a 32-bit machine or a 64-bit JRE for a 64-bit machine.
2. Install a JDBC driver. No special installation setup is required. All you need to do is copy the driver to a local directory, for example, `c:\jdbc`. Note that JDBC drivers (which are Jar files) are platform-independent.
3. Set up the `CLASSPATH` to include the location where the JDBC driver is located. (The application reads the `CLASSPATH` environment variable to locate the JDBC driver.) To access and edit the `CLASSPATH`, do the following: Click **Start | Control Panel | System | Advanced | Environment Variables**. In the Environment Variables dialog that appears, select either the `CLASSPATH` user environment variable or the `CLASSPATH` system environment variable and click the **Edit** button. Add the path to the JDBC driver to the `CLASSPATH`. For example: `CLASSPATH=C:\jdbc\sqljdbc.jar; C:\jdbc\db2jcc.jar;`
4. Log off and then log on again to make the `CLASSPATH` changes active.

Note: Installing the complete IBM DB2 or Oracle client will automatically populate the `CLASSPATH` with the JDBC drivers of the respective packages. For more details, see the summaries below.

Connecting via JDBC

To connect using a JDBC connection, do the following:

1. In the Connect to a Data Source dialog, select **JDBC Connections**. The JDBC Connections pane (*screenshot below*) appears.



2. Select a JDBC driver from the *Driver* dropdown list (all detected drivers, i.e. those in the `CLASSPATH` environment variable, are listed). Enter a connection string in the *Database URL* text box and a user name and password as required. Given below is the connection string syntax for commonly used databases, each with an example connection string.

Oracle `jdbc:oracle:thin:[ user/password]@//[ host][: port]/SID`
 `jdbc:oracle:thin:@//abcd234/ORA11`

IBM DB2 `jdbc:db2://host_name: port/dbname`
 `jdbc:db2://MyDB2: 50000/boz`

MySQL jdbc:mysql://host_name:port/dbname
 jdbc:mysql://MyDB2:3306/moz

MSSQL jdbc:sqlserver://host:port;databasename=name;user=name;password=Pwd
 jdbc:sqlserver://abcd38:1433;databasename=coz
 jdbc:sqlserver://Q5;DatabaseName=coz;SelectMethod=Cursor

PostgreSQL jdbc:postgresql://host:port/database
 jdbc:postgresql://abc993:5432/qanoz

Sybase jdbc:sybase:Tds:host:port/dbname
 jdbc:sybase:Tds:abc12:2048/QUE

3. Click the **Connect** button. The connection to the data source is established. For information about subsequent dialogs, refer to the description of the command being executed.

Step-by-step: MSSQL, MySQL, PostGre, and other non-XML DBs

Given below is a step-by-step guide for connecting to a non-XML DB via JDBC:

1. JDBC JAR files (driver files): Copy the files to any local location. Then add the full path and filename to the Windows `CLASSPATH` variable. For example:
`C:\jdbc\sqljdbc.jar; C:\jdbc\db2jcc.jar; .`
2. Log off and log on to make the `CLASSPATH` changes active.
3. Start XMLSpy and access the Connect to a Data Source dialog.
4. The JDBC Connections pane lists, in the *Driver* dropdown box, the detected JDBC drivers. If the dropdown box is empty, make sure that the file `altovadb.jar` is present in the folder: `C:\Program Files\Altova\Common2012\jar`.
5. Connect to a database as described in the section *Connecting via JDBC* above.

Step-by-step: Oracle

Given below is a step-by-step guide for connecting to an Oracle DB via JDBC. If you don't need the XML and XDB features of the Oracle DB, follow the instructions in the step-by-step guide for non-XML DBs. The installation folder of the Oracle client is indicated in the steps below by the placeholder: `%ORACLE_HOME%`.

1. Install Oracle client software with OCI and ODBC features enabled. If an Oracle client is already installed check if the two jar files below are present:
`%ORACLE_HOME%\LIB\xmlparserv2.jar`
`%ORACLE_HOME%\RDBMS\jlib\xdb.jar`
2. Add the following files to the Windows `CLASSPATH` environment variable:
`%ORACLE_HOME%\jdbc\lib\ojdbc6.jar`


```
%ORACLE_HOME%\LIB\xmlparserv2.jar  
%ORACLE_HOME%\RDBMS\jlib\xdb.jar
```

3. Log off and log on to make the `CLASSPATH` changes active.
4. Start XMLSpy and access the Connect to a Data Source dialog.
5. The JDBC Connections pane lists, in the *Driver* dropdown box, the detected JDBC drivers. If the dropdown box is empty, make sure that the file `altovadb.jar` is present in the folder: `C:\Program Files\Altova\Common2012\jar`.
6. Connect to a database as described in the section *Connecting via JDBC* above.

Step-by-step: IBM DB2

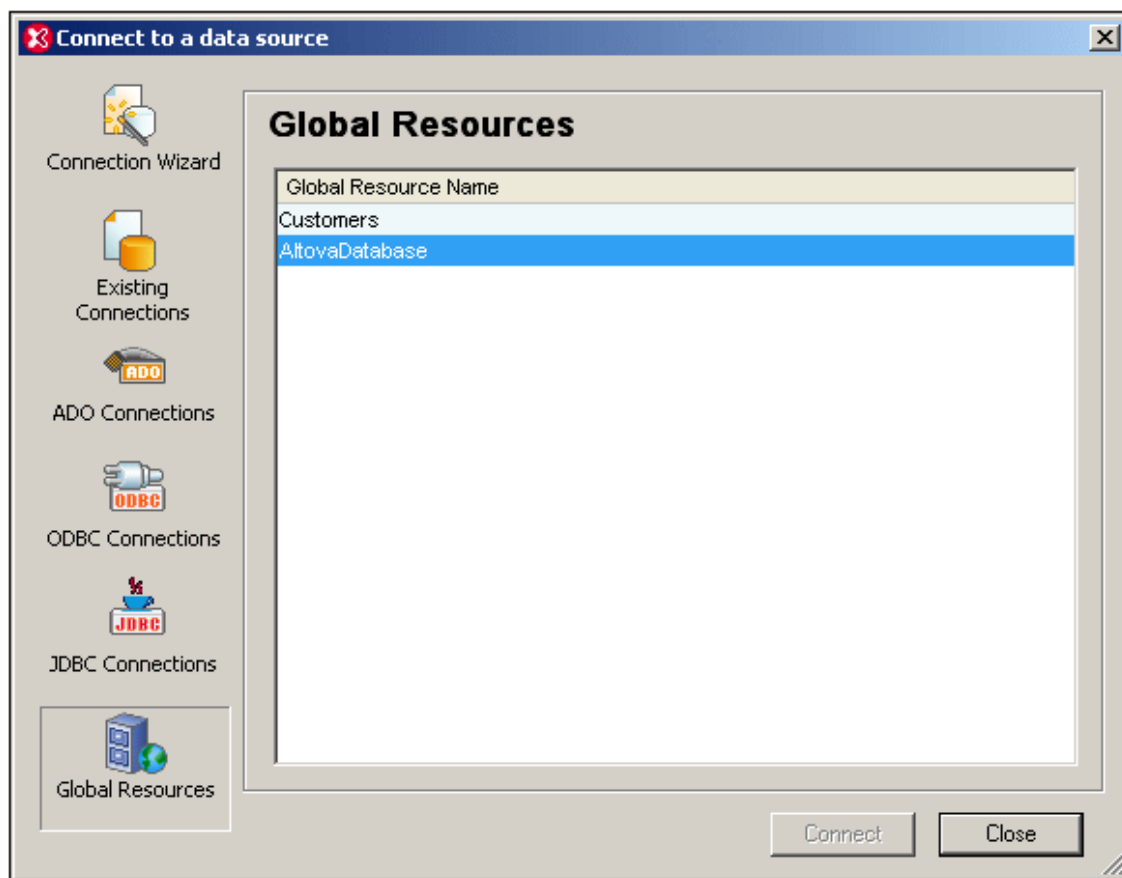
Given below is a step-by-step guide for connecting to an IBM DB2 database via JDBC.

1. If an IBM DB2 client has already been installed, nothing needs to be done since the `CLASSPATH` will have been set by the installation process.
2. If no IBM DB2 client has been installed, add the IBM DB2 JDBC driver jar files `db2jcc.jar` and `db2jcc_license_cu.jar` to the Windows `CLASSPATH`. Log off and log on to make the `CLASSPATH` changes active.
3. Start XMLSpy and access the Connect to a Data Source dialog.
4. The JDBC Connections pane lists, in the *Driver* dropdown box, the detected JDBC drivers. If the dropdown box is empty, make sure that the file `altovadb.jar` is present in the folder: `C:\Program Files\Altova\Common2012\jar`.
5. Connect to a database as described in the section *Connecting via JDBC* above.

Note: When databases are connected to via JDBC, due to insufficient information returned by the drivers: (i) data editing is not possible for tables without a primary key; (ii) Execute for data-editing in SQL Editor will not work.

Global Resources

In the Connect to Data Source dialog, when the **Global Resources** button is selected, the Global Resources pane opens (*screenshot below*), showing all the database-type global resources that are defined in the currently active [Global Resources XML File](#).

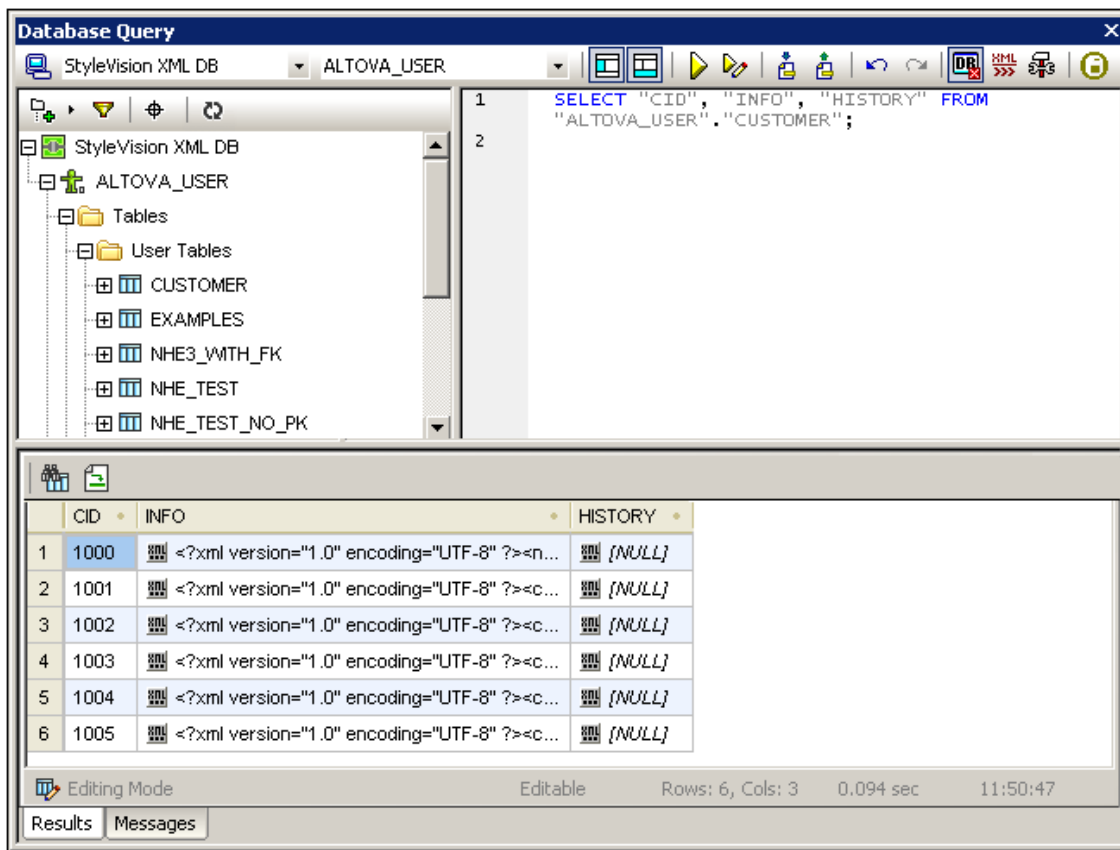


Select the required global resource and click **Connect**. The connection to the database is established. For information about subsequent dialogs, refer to the description of the command being executed.

21.9.2 Query Database

The **Query Database** command opens the Database Query window (*screenshot below*). Once the Query Window is open, its display can be toggled on and off by clicking either the **DB** |

Query Database command or the Query Database toolbar icon .



Overview of the Database Query window

The Database Query window consists of three parts:

- A [Browser pane](#) at top left, which displays connection info and database tables.
- A [Query pane](#) at top right, in which the query is entered.
- A tabbed [Results/Messages pane](#). The Results pane displays the query results in what we call the Result Grid. The Messages pane displays messages about the query execution, including warnings and errors.

The Database Query window has a toolbar at the top. At this point, take note of the two toolbar icons below. The other toolbar icons are described in the section, [Query Pane: Description and Features](#).



Toggles the Browser pane on and off.



Toggles the Results/Messages pane on and off.

Overview of the Query Database mechanism

The Query Database mechanism is as follows. It is described in detail in the sub-sections of this section

1. A [connection to the database is established](#) via the Database Query window. Supported databases include: MS Access 2000 and 2003; Microsoft SQL Server; Oracle; MySQL; Sybase; and IBM DB2.
2. The connected database or parts of it are displayed in the [Browser pane](#), which can be

- configured to suit viewing requirements.
3. A [query](#) written in a syntax appropriate to the database to be queried is entered in the [Query pane](#), and the query is executed.
4. The [results of the query](#) can be viewed through various filters, edited, and saved back to the DB.

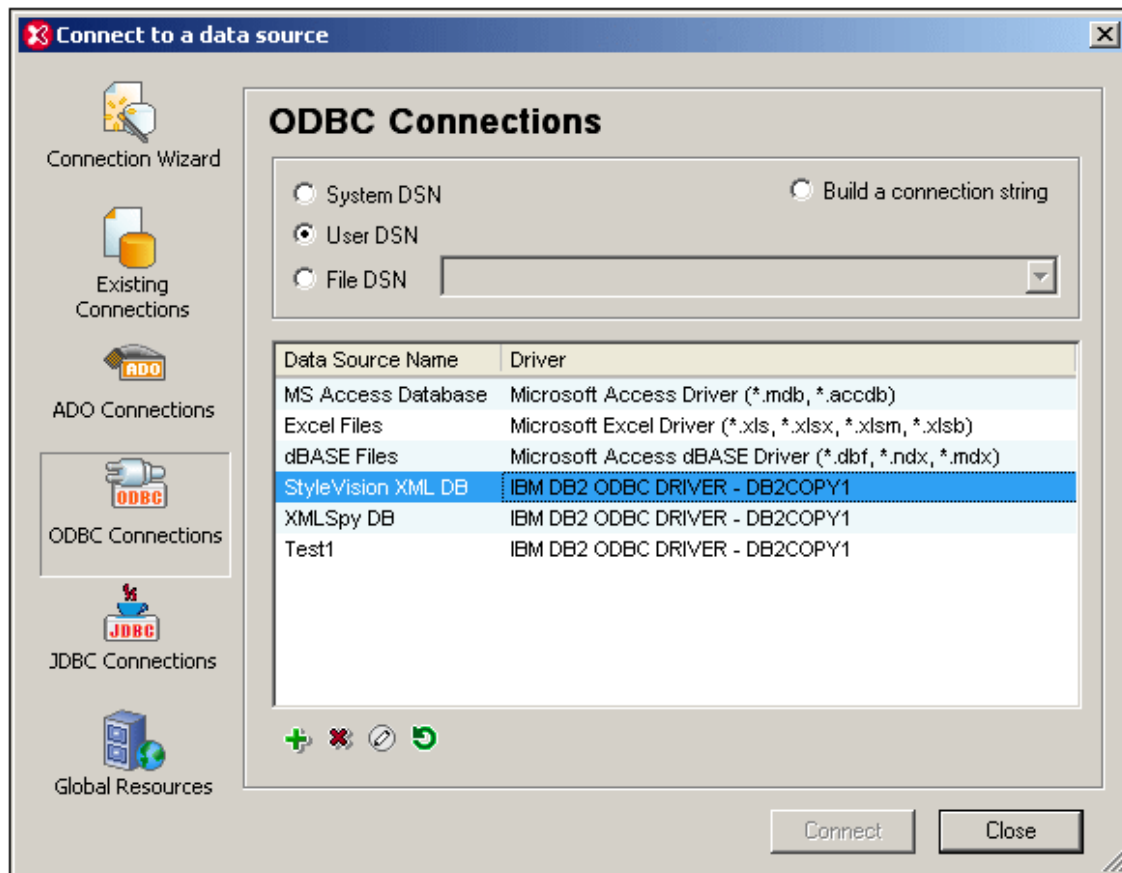
Data Sources

In order to query a database, you have to first connect to the required database. This section describes how to:

- Connect to a database, and
- Select the required data source and root object from among multiple existing connections.

Connecting to a database

When you click the **Query Database** command for the first time in a session (or when no database connection exists), the Quick Connect dialog (*screenshot below*) pops up to enable you to connect to a database. To make connections subsequently, click the Quick Connect icon  in the Database Query window.



How to connect to a database via the Quick Connect dialog is described in the section [Connecting to a Data Source](#).

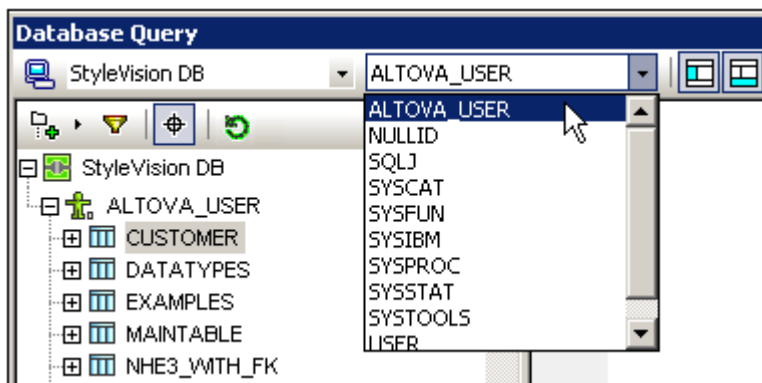
The following databases are supported. After connecting to the database, the available root objects for each of the supported databases is as follows:

Database (natively supported)	Root Object
MS SQL Server 2000, 2005, and 2008	database
MS SQL Server 2005 and 2008	schema
Oracle 9i, 10g, and 11g	schema
MS Access 2003 and 2007	database
MySQL 4.x and 5.x	database
IBM DB2 8.x and 9	schema
Sybase 12	database
IBM DB2 for i 5.4 and i 6.1	schema
PostgreSQL 8.0, 8.1, 8.2, 8.3	database

Note: XMLSpy fully supports the databases listed above. While Altova endeavors to support other ODBC/ADO databases, successful connection and data processing have only been tested with the listed databases.

Selecting the required data source

All the existing connections and the root objects of each are listed, respectively, in two combo boxes in the toolbar of the Database Query window (*screenshot below*).



In the screenshot above, the database with the name `StyleVision DB` has been selected. Of the available root objects for this database, the root object `ALTOVA_USER` has been selected. The database and the root object are then displayed in the Browser pane.

Browser Pane: Viewing the DB Objects

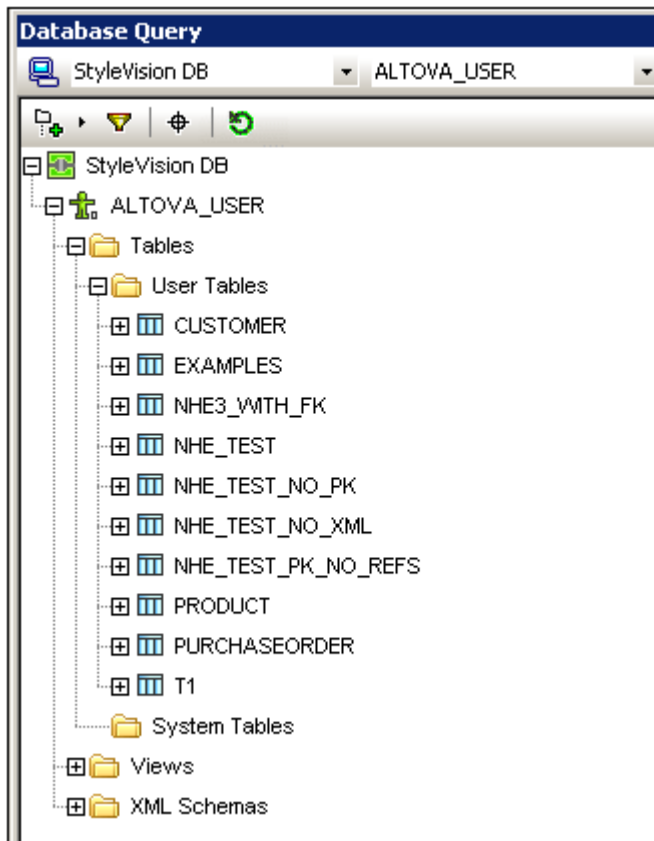
The Browser pane provides an overview of objects in the selected database. This overview includes database constraint information, such as whether a column is a primary or foreign key. In IBM DB2 version 9 databases, the Browser additionally shows registered XML schemas in a separate folder.

This section describes the following:

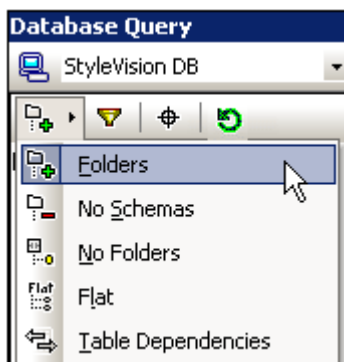
- The [layouts](#) available in the Browser pane.
- [How to filter](#) database objects.
- [How to find](#) database objects.

Browser pane layouts

The default Folders layout displays database objects hierarchically. Depending on the selected object, different context menu options are available when you right-click an item.



To select a layout for the Browser, click the Layout icon in the toolbar of the Browser pane and select the layout from the drop-down list (*screenshot below*). Note that the icon changes with the selected layout.



The available layouts are:

- *Folders*: Organizes database objects into folders based on object type in a hierarchical tree, this is the default setting.
- *No Schemas*: Similar to the Folders layout, except that there are no database schema folders; tables are therefore not categorized by database schema.
- *No Folders*: Displays database objects in a hierarchy without using folders.
- *Flat*: Divides database objects by type in the first hierarchical level. For example, instead of columns being contained in the corresponding table, all columns are displayed in a separate Columns folder.
- *Table Dependencies*: Categorizes tables according to their relationships with other tables. There are categories for tables with foreign keys, tables referenced by foreign keys and tables that have no relationships to other tables.

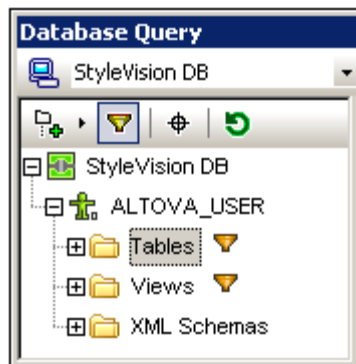
To sort tables into User and System tables, switch to Folders, No Schemas or Flat layout, then right-click the Tables folder and select **Sort into User and System Tables**. The tables are sorted alphabetically in the User Tables and System Tables folders.

Filtering database objects

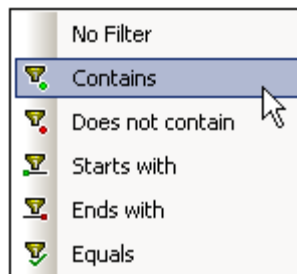
In the Browser pane (in all layouts except No Folders and Table Dependencies), schemas, tables, and views can be filtered by name or part of a name. Objects are filtered as you type in the characters, and filtering is case-insensitive by default.

To filter objects in the Browser, do the following:

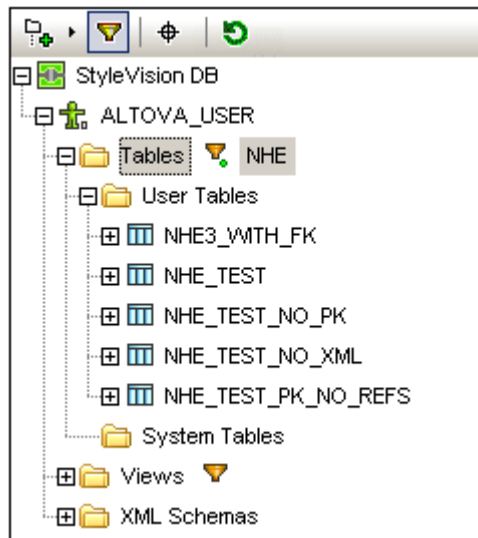
1. Click the Filter Folder Contents icon in the toolbar of the Browser pane. Filter icons appear next to the Tables and Views folders in the currently selected layout (*screenshot below*).



2. Click the filter icon next to the folder you want to filter, and select the filtering option from the popup menu, for example, *Contains*.



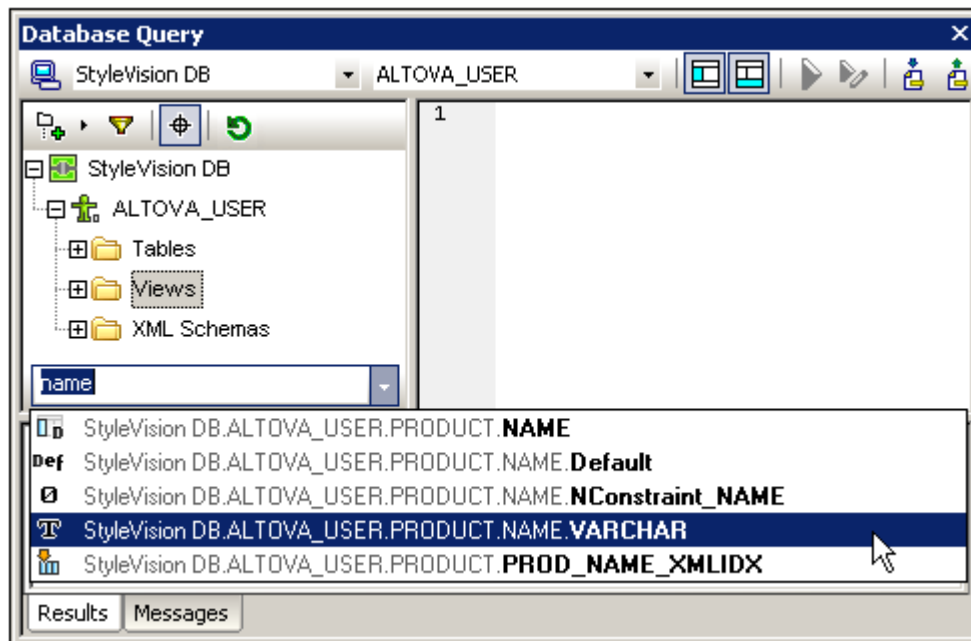
3. In the entry field that appears, enter the filter string (in the screenshot below, the filter string on the *Tables* folder is *NHE*). The filter is applied as you type.



Finding database objects

To find a specific database item by its name, you can use the Browser pane's Object Locator. This works as follows:

1. In the toolbar of the Browser pane, click the Object Locator icon. A drop-down list appears at the bottom of the Browser.
2. Enter the search string in the entry field of this list, for example `name` (screenshot below). Clicking the drop-down arrow displays all objects that contain the search string.



3. Click the object in the list to see it in the Browser.

Query Pane: Description and Features

The Query pane is an intelligent SQL editor for entering queries to the selected database. After entering the query, clicking the Execute command of the Database Query window executes the query and displays the result and execution messages in the [Results/Messages pane](#). How to work with queries is described in the next section, [Query Pane: Working with Queries](#). In this section, we describe the main features of the Query pane:

- SQL Editor icons in the Database Query toolbar
- SQL Editor options
- Auto-completion of SQL statements
- Definition of regions in an SQL script
- Insertion of comments in an SQL script
- Use of bookmarks

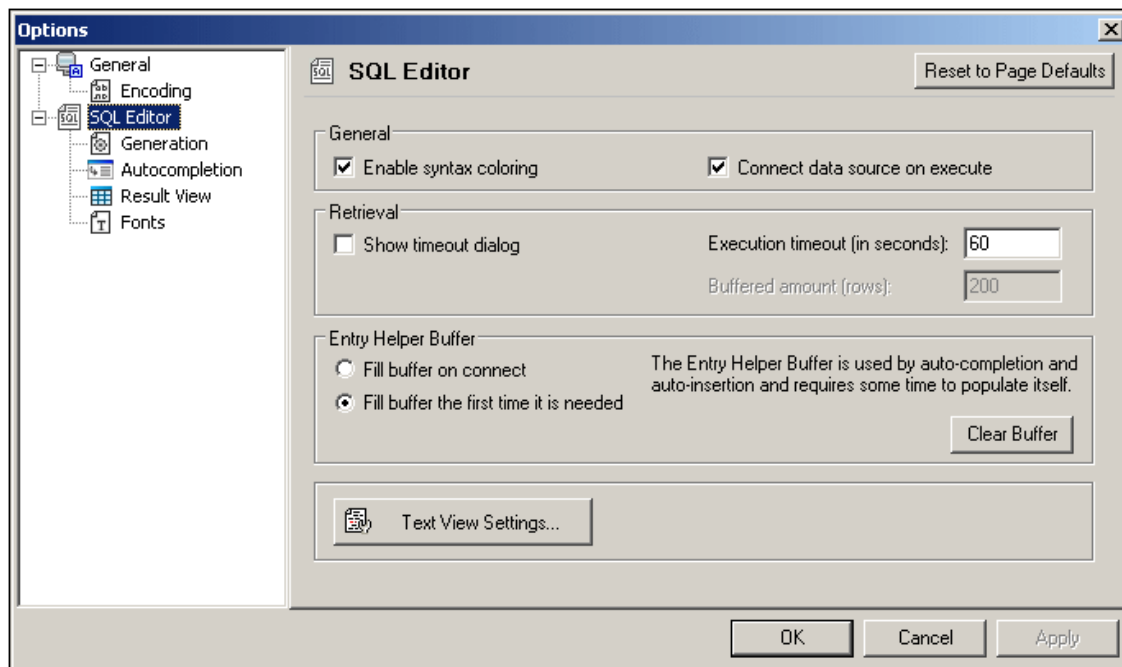
SQL Editor icons in the Database Query toolbar

The following icons in the toolbar of the Database Query window are used when working with the SQL Editor:

	Execute	Executes currently selected SQL statement. If script contains multiple statements and none is selected, then all are executed.
	Execute with Data Editing	Same as for Execute command, except that results (in Results tab) are editable.
	Import SQL File	Opens an SQL file in the SQL Editor.
	Export SQL File	Saves SQL queries to an SQL file.
	Undo	Undoes an unlimited number of edits in SQL Editor.
	Redo	Redoes an unlimited number of edits in SQL Editor.
	Hide DB Query on XML Open	Sets whether the DB Query window should be hidden when an XML document is opened for editing.
	Auto-Commit on XML Save	When an edited XML document is saved in XMLSpy, changes are committed to the DB if this toggle is on. Otherwise, changes have to be explicitly committed in the Results Pane.
	Options	Open the Options dialog of SQL Editor.
	Open SQL Script in DatabaseSpy	Opens the SQL script in Altova's DatabaseSpy product.

Options

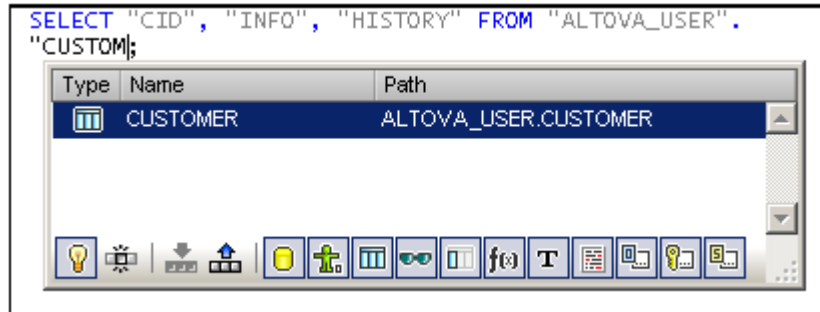
Clicking the **Options** icon in the Database Query toolbar pops up the Options dialog (*screenshot below*). A page of settings can be selected in the left-hand pane, and the options on that page can be selected. Click the **Reset to Page Defaults** button to reset the options on that page to their original settings.



The key settings are as follows:

- General | Encoding:** Options for setting the encoding of new SQL files, of existing SQL files for which the encoding cannot be detected, and for setting the Byte Order Mark (BOM). (If the encoding of existing SQL files can be detected, the files are opened and saved without changing the encoding.)
- SQL Editor:** Options for toggling syntax coloring and data source connections on execution on/off. A timeout can be set for query execution, and a dialog to change the timeout can also be shown if the specified time is exceeded. The buffer for the entry helper information can be filled either on connection to the data source or the first time it is needed. The Text View settings button opens the Text View options window of XMLSpy.
- SQL Editor | SQL Generation:** The application generates SQL statements when you drag objects from the Browser pane into the Query pane. Options for SQL statement generation can be set in the SQL generation tab. Use the *Database* pane to select a database kind and set the statement generation options individually for the different database kinds you are working with. Activating the *Apply to all databases* check box sets the options that are currently selected for all databases. Options include appending semi-colons to statements and surrounding identifiers with escape characters. When the *Append semicolons to statement end* check box is activated, a semicolon is appended when you generate an SQL statement in the SQL Editor. Note that editing of data in Oracle databases and IBM iSeries and DB2 databases via a JDBC connection is possible only if this check box is unchecked.
- SQL Editor | Auto-completion:** The Auto-Completion feature works by suggesting, while you type, relevant entries from various SQL syntax categories. It is available for the following databases: MS SQL Server 2000, 2005, and 2008, MS Access 2003 and 2007, and IBM DB2 v.9. When the Auto-Completion option is switched on, the Auto-Completion window (*screenshot below*) appears, containing suggestions for auto-completion. Select the required entry to insert it. You can define whether the autocompletion popup should be triggered automatically after a delay which you can set in the *Triggering Auto-completion* pane, or if the popup has to be invoked manually. You can also select the keys to be used to insert the selected completion. The SQL Editor can intelligently suggest autocompletion entries based on language statistics. If this feature is activated, items that are frequently used appear on top of the list of suggested

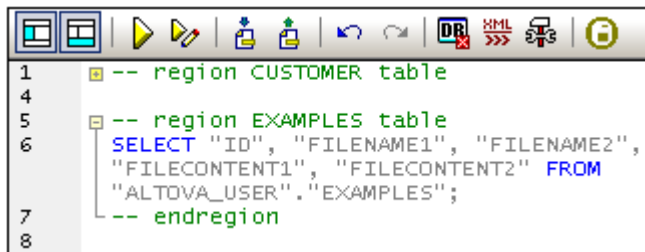
entries. In the Auto-completion window (*screenshot below*) itself, note the buttons at the bottom of the window. The *Context-Sensitive Suggestion* button sets whether only entries that are relevant to the context are displayed or all possible entries with that spelling. The *Single Mode* button enables you to click a category button to select only that category. The *Set All Categories* button selects all categories. You can then deselect a category by clicking its button. The *Clear All Categories* button de-selects all categories. The other buttons are the various category buttons.



- **SQL Editor | Result View:** Options to configure the Result tab.
- **SQL Editor | Fonts:** Options for setting the font style of the text in the Text Editor and in the Result View.

Definition of regions in an SQL script

Regions are sections in SQL scripts that are marked and declared to be a unit. Regions can be collapsed and expanded to hide or display parts of the script. It is also possible to nest regions within other regions. Regions are delimited by `--region` and `--endregion` comments, respectively, before and after the region. Regions can optionally be given a name, which is entered after the `-- region` delimiter (*see screenshot below*).



To insert a region, select the statement/s to be made into a region, right-click, and select **Insert Region**. The expandable/collapsible region is created. Add a name if you wish. In the screenshot above, also notice the line-numbering. To remove a region, delete the two `--region` and `--endregion` delimiters.

Insertion of comments in an SQL script

Text in an SQL script can be commented out. These portions of the script are skipped when the script is executed.

- To comment out a block, mark the block, right-click, and select **Insert/Remove Block Comment**. To remove the block comment, mark the comment, right-click and select **Insert/Remove Block Comment**.
- To comment out a line or part of a line, place the cursor at the point where the line comment should start, right-click, and select **Insert/Remove Line Comment**. To remove the line comment, mark the comment, right-click and select **Insert/Remove Line Comment**.

Use of bookmarks

Bookmarks can be inserted at specific lines, and you can then navigate through the bookmarks in the document. To insert a bookmark, place the cursor in the line to be bookmarked, right-click, and select **Insert/Remove Bookmark**. To go to the next or previous bookmark, right-click, and select **Go to Next Bookmark** or **Go to Previous Bookmark**, respectively. To remove a bookmark, place the cursor in the line for which the bookmark is to be removed, right-click, and select **Insert/Remove Bookmark**. To remove all bookmarks, right-click, and select **Remove All Bookmarks**.

Query Pane: Working with Queries

After connecting to a database, an SQL script can be entered in the SQL Editor and executed. This section describes:

- How an SQL script is entered in the SQL Editor.
- How the script is executed in the Database Query window.

The following icons are referred to in this section:



Execute Query Executes currently selected SQL statement. If script contains multiple statements and none is selected, then all are executed.



Execute for Data Editing Same as for Execute command, except that results (in Results tab) are editable.



Import SQL File Opens an SQL file in the SQL Editor.

Creating SQL statements and scripts in the SQL Editor

The following GUI methods can be used to create SQL statements or scripts:

- *Drag and drop*: Drag an object from the Browser pane into the SQL Editor. An SQL statement is generated to query the database for that object.
- *Context menu*: Right-click an object in the Browser pane and select **Show in SQL Editor | Select**.
- *Manual entry*: Type SQL statements directly in SQL Editor. The Auto-completion feature can help with editing.
- *Import an SQL script*: Click the **Import SQL File** icon in the toolbar of the Database Query window.

Executing SQL statements

If the SQL script in the SQL Editor has more than one SQL statement, select the statement to execute and click either the **Execute** icon or **Execute with Data Editing** icon in the toolbar of the Database Query window. If no statement in the SQL script is selected, then all the statements in the script are executed. The database data is retrieved and displayed as a grid in the [Results tab](#). If **Execute with Data Editing** was selected, then the retrieved data in the Result Grid [can be edited](#). Messages about the execution are displayed in the [Messages tab](#).

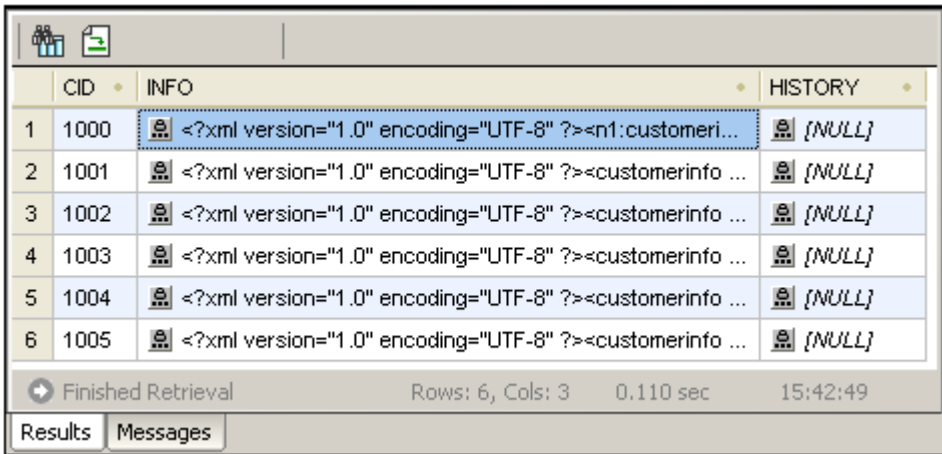
Results and Messages

The Results/Messages pane has two tabs:

- The [Results tab](#) shows the data that is retrieved by the query.
- The [Messages tab](#) shows messages about the query execution.

Results tab

The data retrieved by the query is displayed in the form of a grid in the Results tab (*screenshot below*). When the query results have been generated using the **Execute Query** command, the XML documents in the Results tab are indicated with the XML icon  (*screenshot below*). If the **Execute for Data Editing** command was used, XML documents are shown with the Editable XML icon .



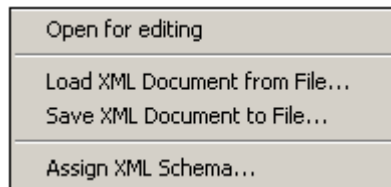
	CID	INFO	HISTORY
1	1000	 <?xml version="1.0" encoding="UTF-8" ?><n1:customer...	 [NULL]
2	1001	 <?xml version="1.0" encoding="UTF-8" ?><customerinfo ...	 [NULL]
3	1002	 <?xml version="1.0" encoding="UTF-8" ?><customerinfo ...	 [NULL]
4	1003	 <?xml version="1.0" encoding="UTF-8" ?><customerinfo ...	 [NULL]
5	1004	 <?xml version="1.0" encoding="UTF-8" ?><customerinfo ...	 [NULL]
6	1005	 <?xml version="1.0" encoding="UTF-8" ?><customerinfo ...	 [NULL]

Finished Retrieval Rows: 6, Cols: 3 0.110 sec 15:42:49

Results Messages

The following operations can be carried out in the Results tab, via the context menu that pops up when you right-click in the appropriate location in the Results tab:

- *Sorting on a column:* Right-click anywhere in the column on which the records are to be sorted, then select **Sorting | Ascending/Descending/Restore Default**.
- *Copying to the clipboard:* This consists of two steps: (i) selecting the data range; and (ii) copying the selection. Data can be selected in several ways: (i) by clicking a column header or row number to select the column or row, respectively; (ii) selecting individual cells (use the **Shift** and/or **Ctrl** keys to select multiple cells); (iii) right-clicking a cell, and selecting **Selection | Row/Column/All**. After making the selection, right-click, and select **Copy Selected Cells**. This copies the selection to the clipboard, from where it can be pasted into another application.
- *Appending a new row:* If the query was executed for editing, right-click anywhere in the Results pane to access the **Append row** command.
- *Deleting a row:* If the query was executed for editing, right-click anywhere in a row to access the **Delete row** command.
- *Editing records:* If the query was executed for editing, individual fields can be edited. To commit changes, click the **Commit** button in the toolbar of the Results tab.
- *Editing XML records:* If the query was executed for editing and an editable field is an XML field (only IBM DB2 XML databases are currently supported), clicking the Editable XML icon  in the Result Grid opens the Edit XML menu (*screenshot below*). An XML field can also be opened for data editing by right-clicking the XML field in the Folders pane and selecting the command **Edit Data**.



The **Open for Editing** command opens the XML document in an XMLSpy window, and the Editable XML icon changes to . When this document is saved and if the Auto-Commit XML Changes icon  in the Query Database toolbar was selected when the document was opened, the changes to the XML document are committed automatically to the database. Otherwise, saved changes will have to be committed using the Commit button of the Results pane. (Note that to toggle between the XML document window and the Database Query window, you must click the **DB | Query Database** command.) The **Load XML Document from File** command loads an external XML document to the selected field in the database. The **Save XML Document to File** saves the XML document in the selected database field to a file location you choose. The Assign XML Schema command pops up the [Choose XML Schema dialog](#), in which you can select an XML Schema to assign to the XML document. This assignment is saved to the database. XML Schema assignment is explained in more detail in the section, [IBM DB2 | Assign XML Schema](#).

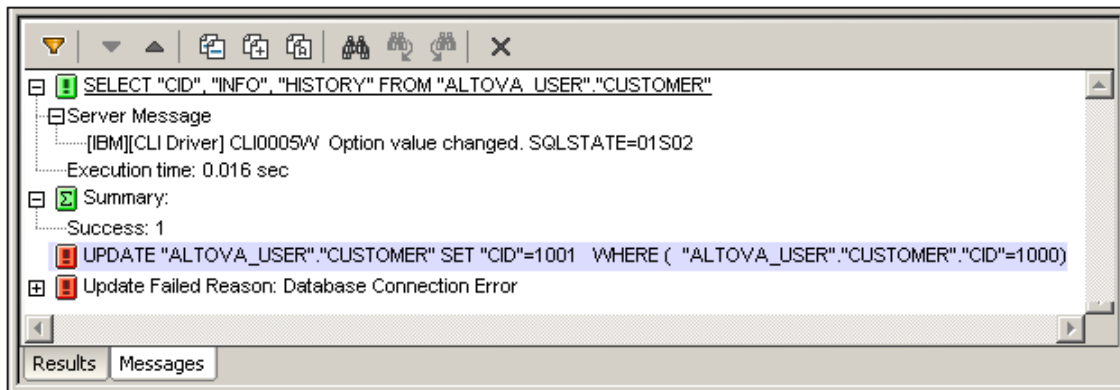
- *Set NULL, Set default, Undo changes for this cell:* If the query was executed for editing, right-clicking in a cell provides access to commands that enable you to set a `NULL` value or, if defined, a column default value for that cell. Changes made to a cell can be undone with the **Undo changes for this cell** command; the current edited value is replaced by the value currently in the DB.

The Results tab has the following toolbar icons:

	Go to Statement	Highlights the statement in the SQL Editor that produced the current result.
	Find	Finds text in the Results pane. XML document content is also searched.
	Add New Line	Adds a new row to the Result Grid.
	Delete Row	Deletes the current row in the Result Grid.
	Undo Changes to Result Grid	Undoes all changes to the Result Grid.
	Commit	Commits changes made in the Result Grid to the database.

Messages tab

The Messages tab provides information on the previously executed SQL statement and reports errors or warning messages.



The toolbar of the Messages tab contains icons that enable you to customize the view, navigate it, and copy messages to the clipboard. The **Filter** icon enables the display of particular types of messages to be toggled on or off. The **Next** and **Previous** icons move the selection down and up the list, respectively. Messages can also be copied with or without their child components to the clipboard, enabling them to be pasted in documents. The **Find** function enables you to specify a search term and then search up or down the listing for this term. Finally, the **Clear** icon clears the contents of the Messages pane.

Note: These toolbar icon commands are also available as context menu commands.

21.9.3 IBM DB2

The **IBM DB2** menu item rolls out a submenu containing commands (i) to register and unregister schemas with an IBM DB2 database ([Manage XML Schemas](#)), and (ii) to assign schemas for XML file validation ([Assign XML Schema](#)).

Both these mechanisms require that you connect to the required IBM DB2 database. How to connect to the database is described in the section [Connecting to a Data Source](#). In this section the focus is on how to manage schemas in an IBM DB2 database and how to assign XML Schemas to a DB XML file.

Note: The Result Grid of the [Database Query window](#) provides important functionality for working with XML files in IBM DB2 databases. This functionality includes the ability to open files for editing, loading XML files into a DB XML files, saving DB XML files externally, and assigning XML Schemas to DB XML files.

Manage XML Schemas

The **Manage XML Schemas** feature enables schemas to be added to and dropped from individual database schemas in an IBM DB2 database. To manage schemas, you have to do the following:

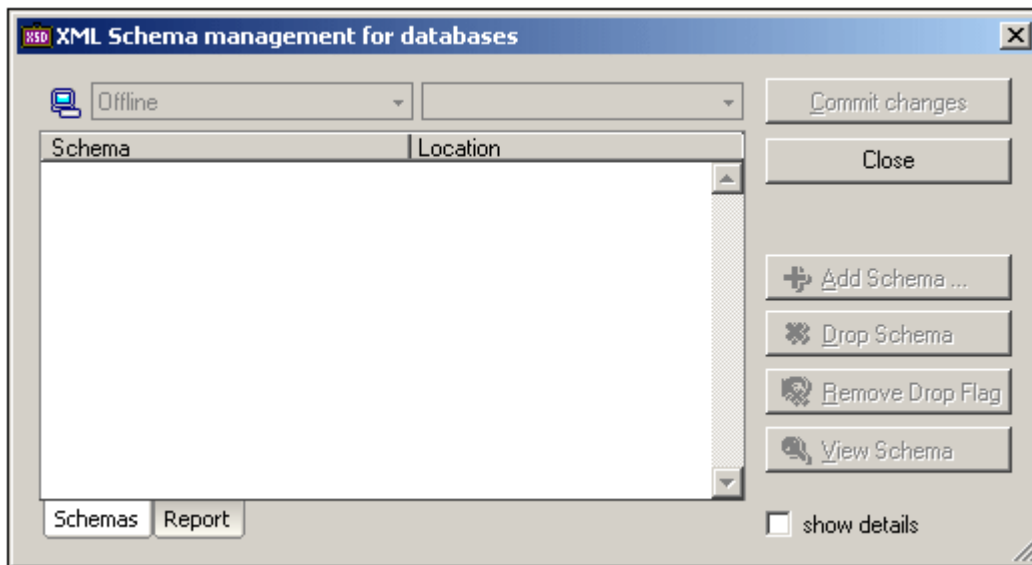
- Connect to the IBM DB2 database
- Select the database schema for which XML Schemas need to be added or dropped
- Carry out the schema management actions.

These steps are described in detail below.

Connecting to the IBM DB2 database

Clicking the **Manage XML Schemas** command pops up the XML Schema Management for

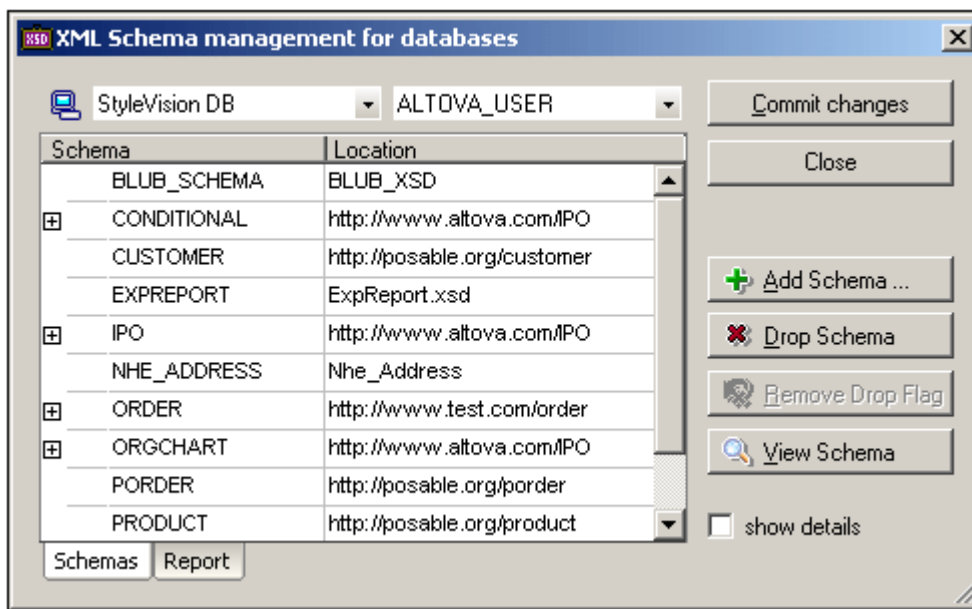
Databases dialog (*screenshot below*).



The first thing to do if there is no connection to the required database is to connect to it. If a connection already exists, it appears in the Database combo box. To start the connection process, click the Quick Connect icon  in the dialog. This pops up the Quick Connect dialog, through which you can make the connection to the database. How to use the Quick Connect dialog is explained in the section [DB Menu | Connecting to a Data Source](#).

Displaying the list of XML Schemas

After the connection to the IBM DB2 database has been established, the database is listed in the combo box at left (*see screenshot below*). If more than one connection is currently open, you can select the required database in this combo box. In the screenshot below, the StyleVision DB database is selected.



The combo box at right lists all the database schemas of the currently selected IBM DB2 database. When a database schema is selected in this combo box, all the XML Schemas

registered for the selected database schema are displayed in the main pane. In the screenshot above, all the XML Schemas registered with the `Altova User` database schema are listed, together with their locations. Checking the Show Details check box causes additional information columns to be displayed in the main pane.

Managing the XML Schemas

The list of schemas in the main pane represents the schemas registered for the selected database schema. After the list of XML Schemas is displayed, you can add schemas to the list or drop (delete) schemas from the list.

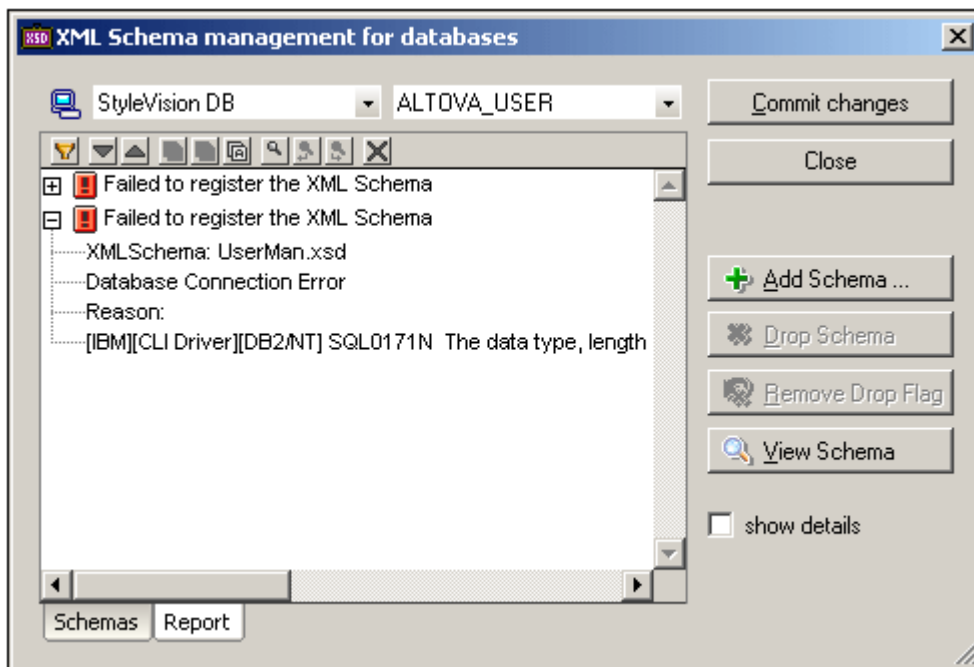
To add a schema, click the **Add** button, browse for the required schema file, and select it. The selected schema file is added to the list in the main pane. Clicking the **Commit Changes** button registers the newly added schema with the database schema.

To drop a schema, select the schema in the main pane and click the **Drop Schema** button. A Drop Flag is assigned to the schema, indicating that it is scheduled for dropping when changes are next committed. The Drop Flag can be removed by selecting the flagged schema and clicking the **Remove Drop Flag** button. When the **Commit Changes** button is clicked, all schemas that have been flagged for dropping will be unregistered from the database schema.

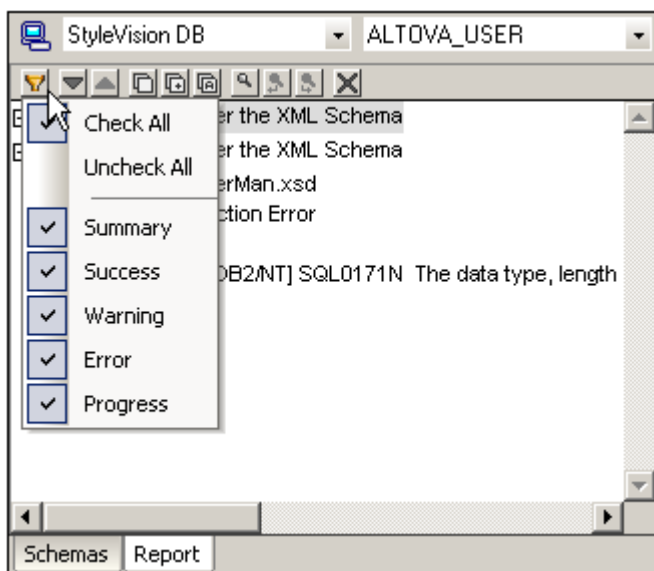
Clicking the View Schema button opens the schema in XMLSpy. To close the XML Schema Management dialog, click the **Close** button.

Reports

When the **Commit Changes** button is clicked, the database is modified according to the changes you have made. A report of the Commit action is displayed in the Report pane (*screenshot below*), enabling you to evaluate whether the success of the action and to debug possible errors. Each subsequent report is displayed below the previous report.



The report pane has a toolbar containing icons that enable you to customize the display of the report listing, navigate the listing, copy report messages, search for text, and clear the pane (*see screenshot below*).



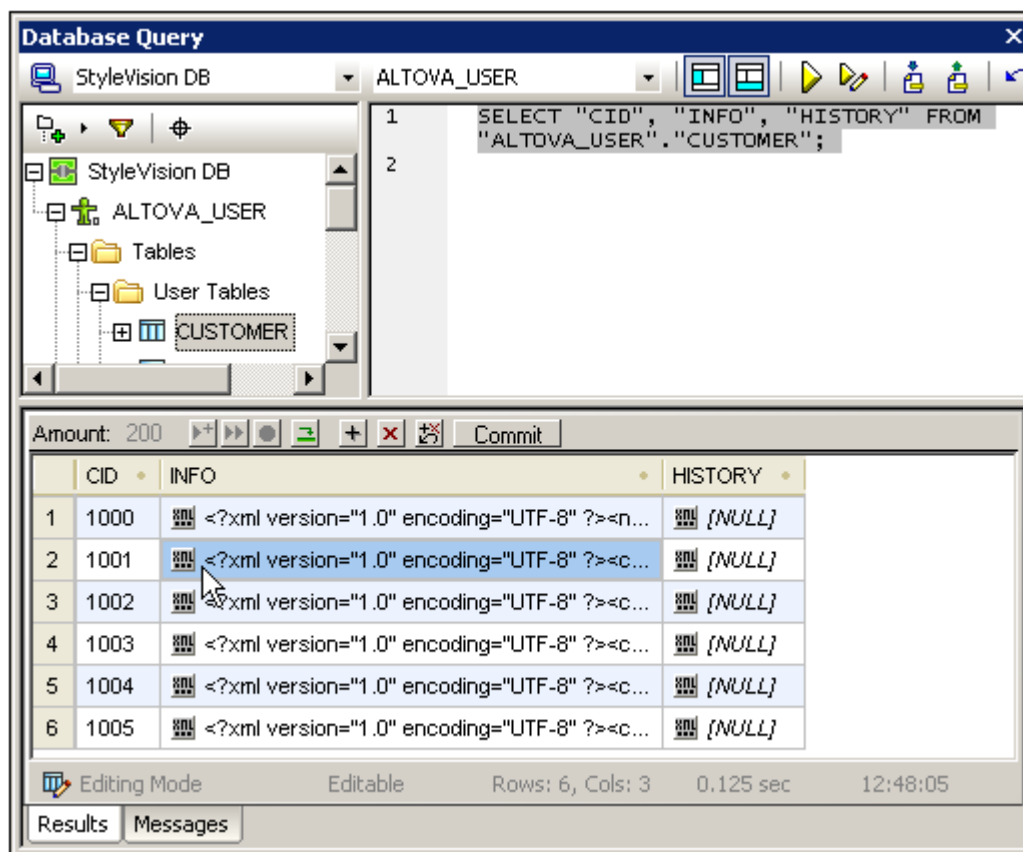
The **Filter** icon enables the display of particular types of messages to be toggled on or off. The **Next** and **Previous** icons move the selection down and up the list, respectively. Messages can also be copied with or without their child components to the clipboard, enabling them to be pasted in documents. The **Find** function enables you to specify a search term and then search up or down the listing for this term. Finally, the **Clear** icon clears the contents of the Report pane.

Assign XML Schema

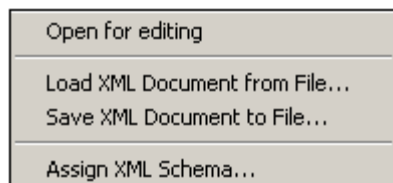
The Assign XML Schema assigns a schema to an XML file opened for editing via the Result Grid of the Database Query window. After the assignment is made, the XML file can be validated against the assigned schema. The assignment is written to the DB when the XML file is saved in XMLSpy.

Opening a DB XML file for editing

In the Database Query window, when a query is addressed to an XML DB and the query is executed for data editing, the Result Grid at the bottom of the Database Query window provides access to the XML files in the database so these can be edited (*see screenshot below*).



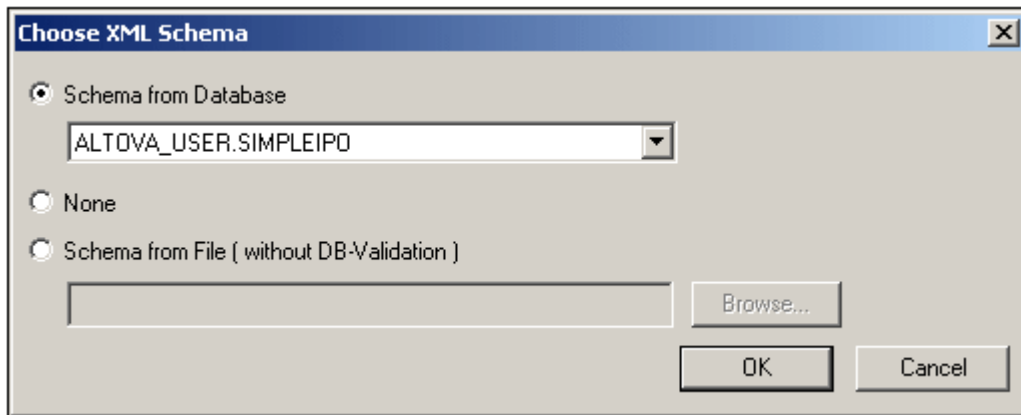
Clicking the XML icon  pops up the following menu.



Selecting the **Open for Editing** command opens the XML document in XMLSpy, where it can be edited.

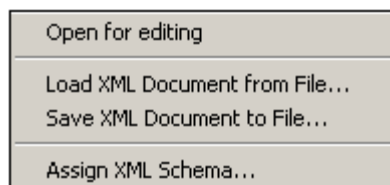
Assigning a schema to the DB XML file

It is when the DB XML file is opened for editing in XMLSpy that the **IBM DB2 | Assign XML Schema** command is enabled. With the XML document active in XMLSpy, clicking the **Assign XML Schema** command pops up the Choose XML Schema dialog (*screenshot below*).



A schema can be selected from among those stored in the database (these are listed in the dropdown list of the Schema from Database combo box), or from among external files that can be browsed. Clicking **OK** assigns the schema to the XML file. Note that the assignment is not written into the XML file. When the XML file is saved in XMLSpy—and if the Auto-Commit XML changes icon  in the Query Database toolbar was selected when the document was opened—then the schema assignment is saved to the database. Note that the schema assignment is written to the database—and not to the XML file.

Note: The Edit XML menu in the Result Grid of the Database Query window also has an **Assign XML Schema** command (see screenshot below), which also assigns a schema to the DB XML file.



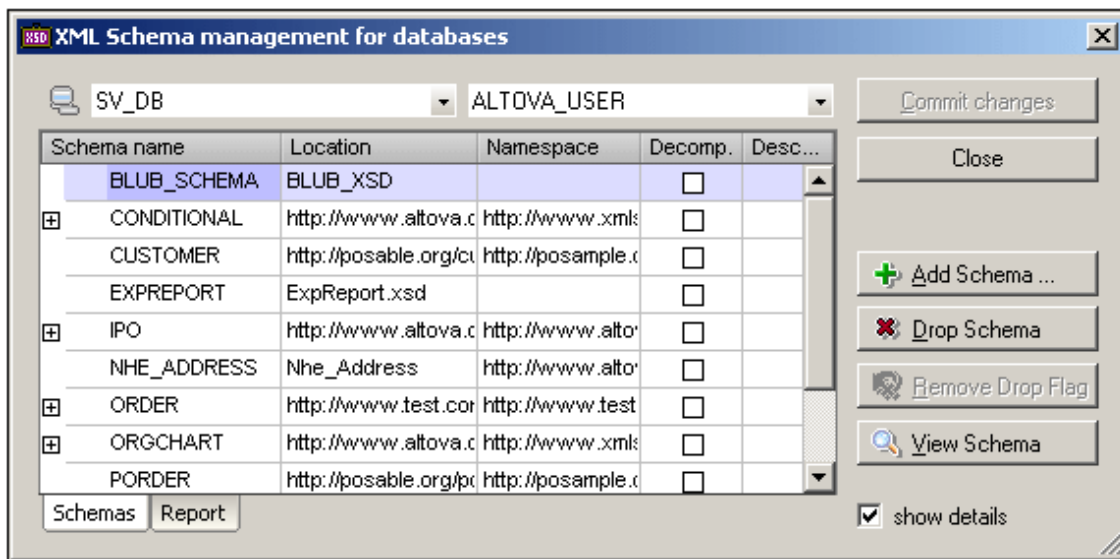
The difference between the two Assign XML Schema commands is that the command in the **DB | IBM DB2** menu enables you to assign an XML Schema while you are editing the XML file thereby allowing you to change schema assignments while editing the XML document and to validate the XML document immediately.

21.9.4 SQL Server

The **SQL Server** menu item rolls out a submenu containing the Manage XML Schemas command.

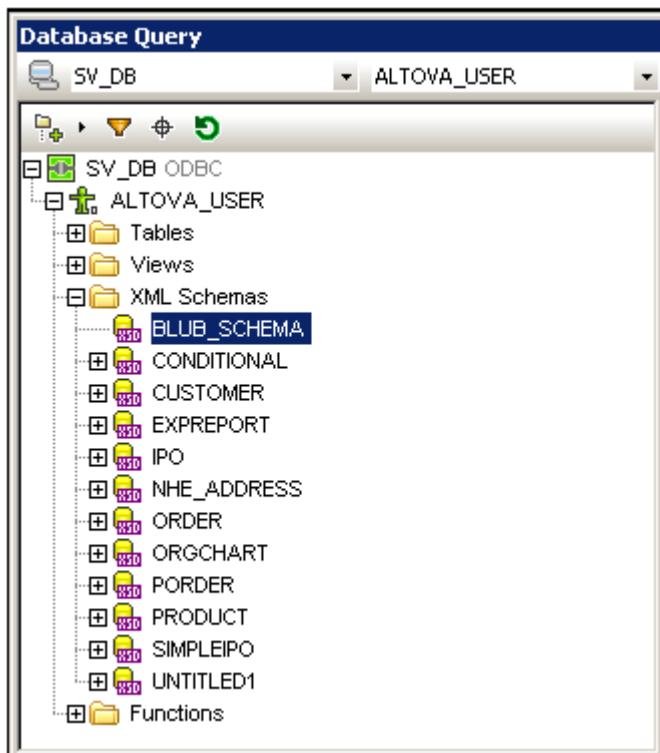
Manage XML Schemas

XML Schema management for databases enables you to add and delete XML Schemas from the schema repository of an XML database. After connecting to the database, XMLSpy provides the XML Schema Management for Databases dialog, in which XML Schemas can be managed.



The dialog box provides a Quick Connect  icon which calls the [Quick Connect wizard](#) to connect to a data source. If more than one connection currently exists, the required connection can be selected from the combo box on the left-hand side. The required root object can then be selected from the right-hand side combo box. All the XML Schemas currently in the repository for that root object are displayed in the dialog box. The name, location, and namespace of each schema are listed, as well as the option for decomposition..

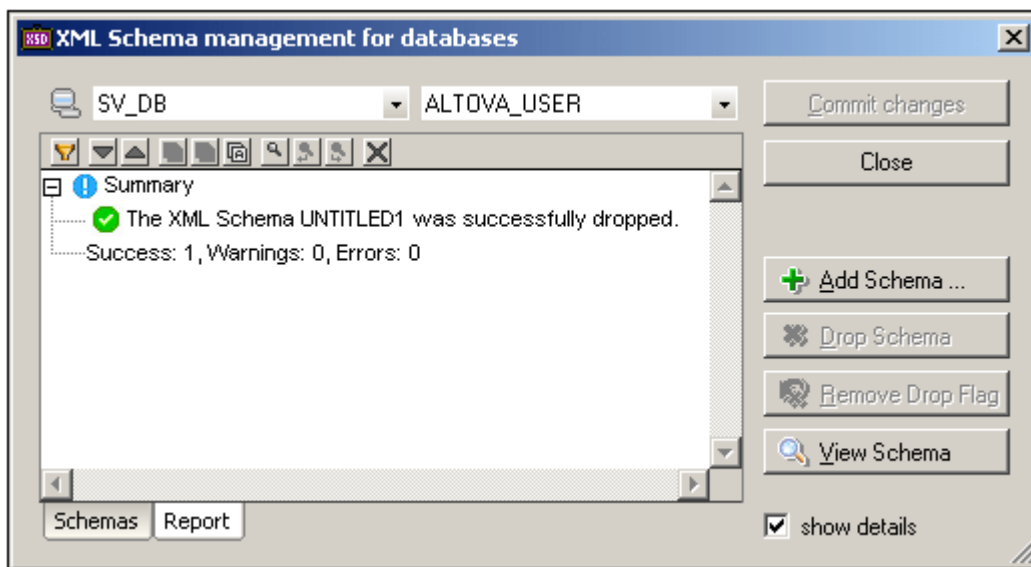
Note that the stored schemas can also be viewed in the Database Query window (*screenshot below*), but they cannot be managed there. To manage schemas, use the XML Schema Management for Databases dialog.



In the XML Schema Management dialog you can do the following:

- Add a schema using the **Add Schema** button. The selected schema will be appended to the list and marked for addition.
- Mark schemas in the list for deletion with the **Drop Schema** button. The Drop flag can be removed with the **Remove Drop Flag** button.
- Open a selected schema in Schema View by clicking the **View Schema** button.
- Commit the addition and drop (deletion) changes with the **Commit Changes** button.

After changes have been committed, a report of the commit action can be viewed in the Report tab (*screenshot below*).



21.9.5 Oracle XML DB

XMLSpy allows you to connect to and query Oracle XML Db databases.

The following database functions are supported:

- Add (and register) an XML schema to the Oracle XML Db. The Oracle XML DB client must be installed for you to be able to register XML schemas through XMLSpy.
- Open and delete schemas
- Query the database using XPath statements (DBUri)
- Browse XML documents (using WebDAV)
- Create an XML document based on a schema saved in the database

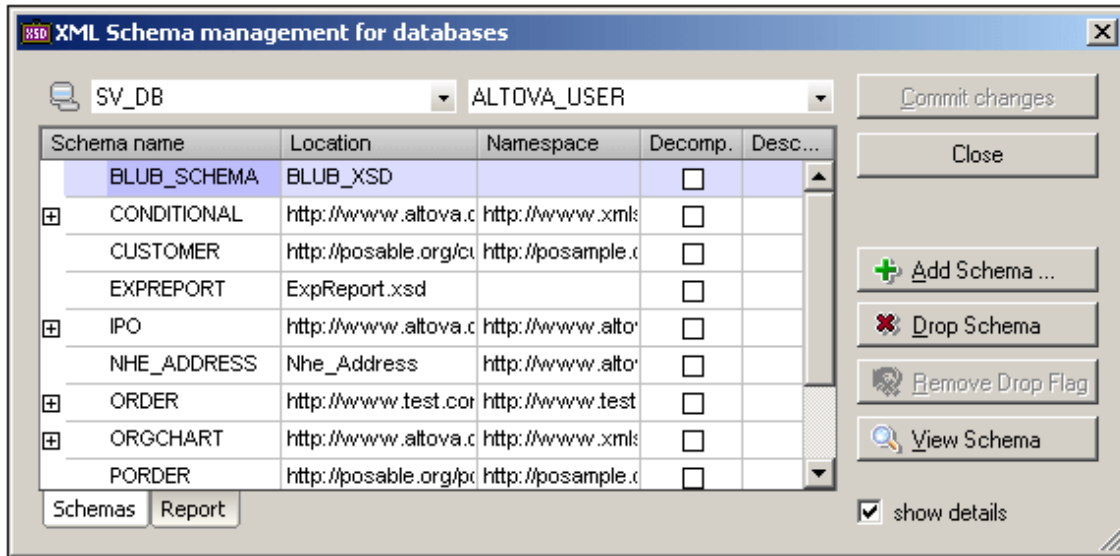
General installation process:

- Download and install XMLSpy
- Install Oracle server (if necessary)
- Create an Oracle database

An Oracle XML DB Demo installation manual is provided at http://otn.oracle.com/sample_code/tech/xml/xmlldb/XDBBasicDemo.pdf. The document describes how to install the Oracle software as well as how to configure it to work with XMLSpy.

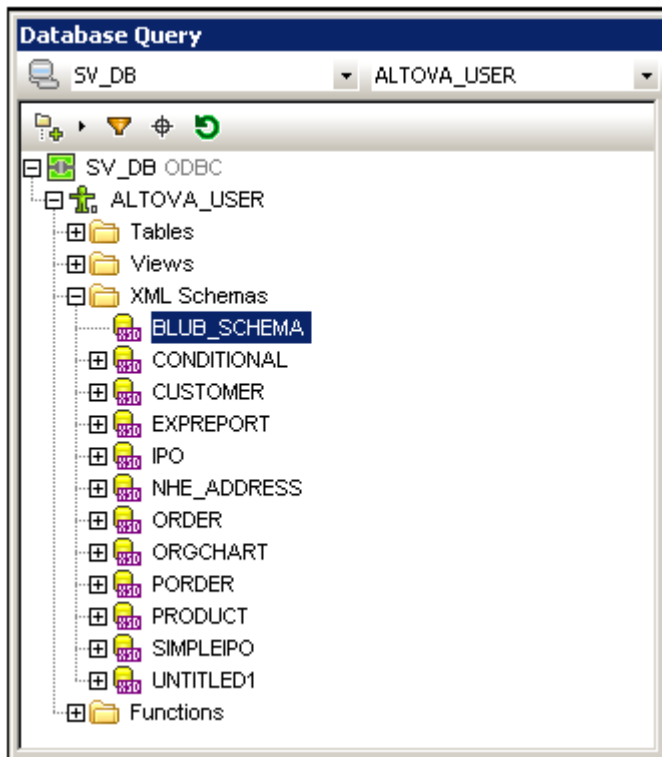
Manage XML Schemas

XML Schema management for databases enables you to add and delete XML Schemas from the schema repository of an XML database. After connecting to the database, XMLSpy provides the XML Schema Management for Databases dialog, in which XML Schemas can be managed.



The dialog box provides a Quick Connect  icon which calls the [Quick Connect wizard](#) to connect to a data source. If more than one connection currently exists, the required connection can be selected from the combo box on the left-hand side. The required root object can then be selected from the right-hand side combo box. All the XML Schemas currently in the repository for that root object are displayed in the dialog box. The name, location, and namespace of each schema are listed, as well as the option for decomposition..

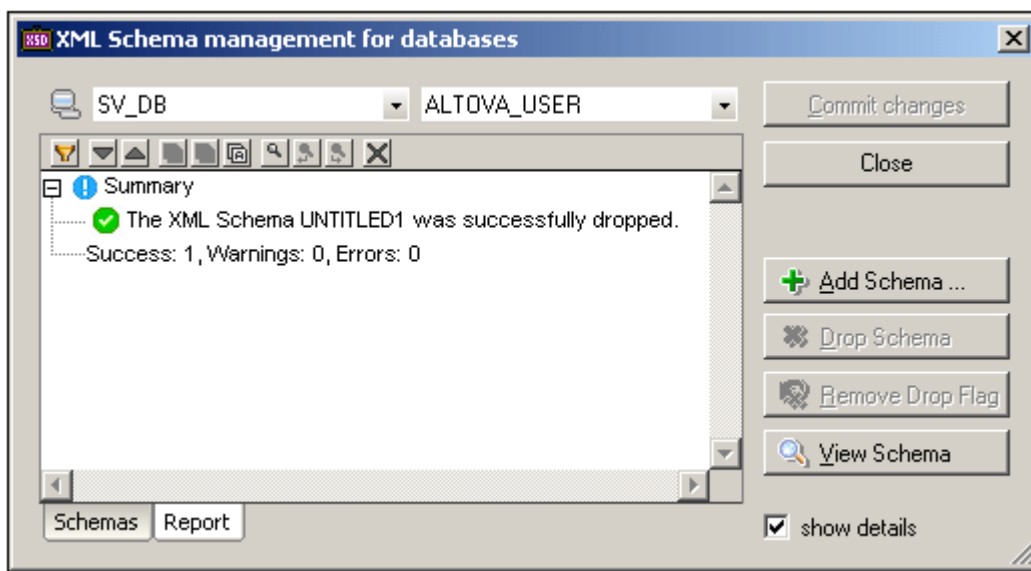
Note that the stored schemas can also be viewed in the Database Query window (*screenshot below*), but they cannot be managed there. To manage schemas, use the XML Schema Management for Databases dialog.



In the XML Schema Management dialog you can do the following:

- Add a schema using the **Add Schema** button. The selected schema will be appended to the list and marked for addition.
- Mark schemas in the list for deletion with the **Drop Schema** button. The Drop flag can be removed with the **Remove Drop Flag** button.
- Open a selected schema in Schema View by clicking the **View Schema** button.
- Commit the addition and drop (deletion) changes with the **Commit Changes** button.

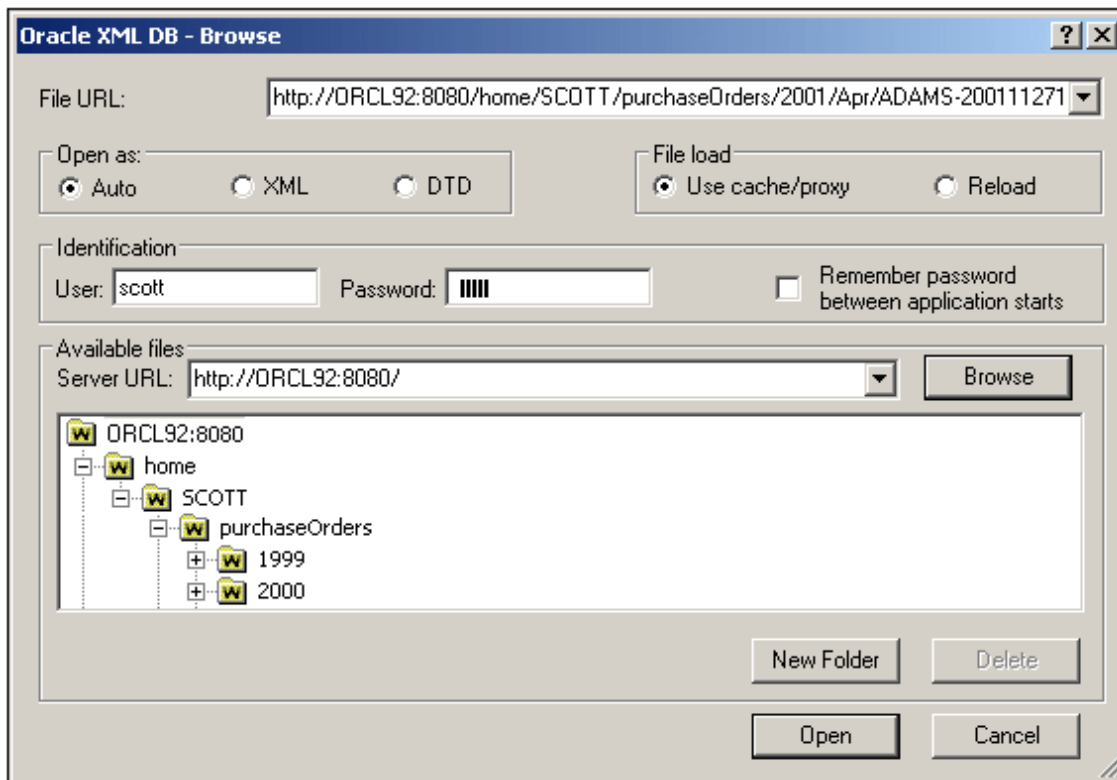
After changes have been committed, a report of the commit action can be viewed in the Report tab (*screenshot below*).



Browse Oracle XML documents



This command allows you to browse the XML documents available on your server. The server details are automatically filled in if you previously queried the database or listed schemas. If this is not the case, then you have to enter them manually.



Use the tree view to find specific XML files. Double clicking a file in the tree view opens it. You

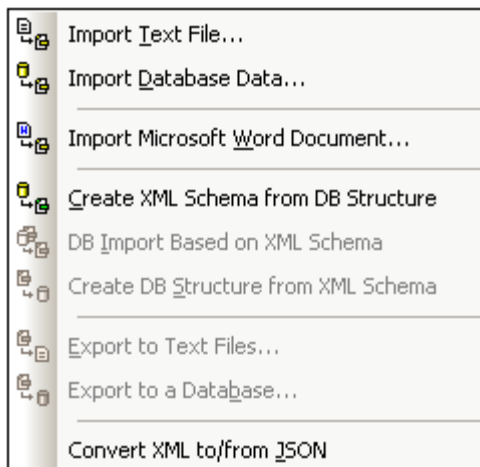
can also click a file and click **Open** to achieve the same thing. The **New Folder** button adds a new folder, the **Delete** button deletes the currently selected XML file.

21.10 Convert Menu

XMLSpy provides powerful data exchange functions that allow you to:

- Import and export text, word processor, database, and XML files.
- [Import database](#) data based on an existing XML Schema.
- [Create an XML Schema](#) based on the structure of an existing database.
- Create a [database structure](#), based on an existing XML schema

These functions are available in the **Convert** menu (*screenshot below*).

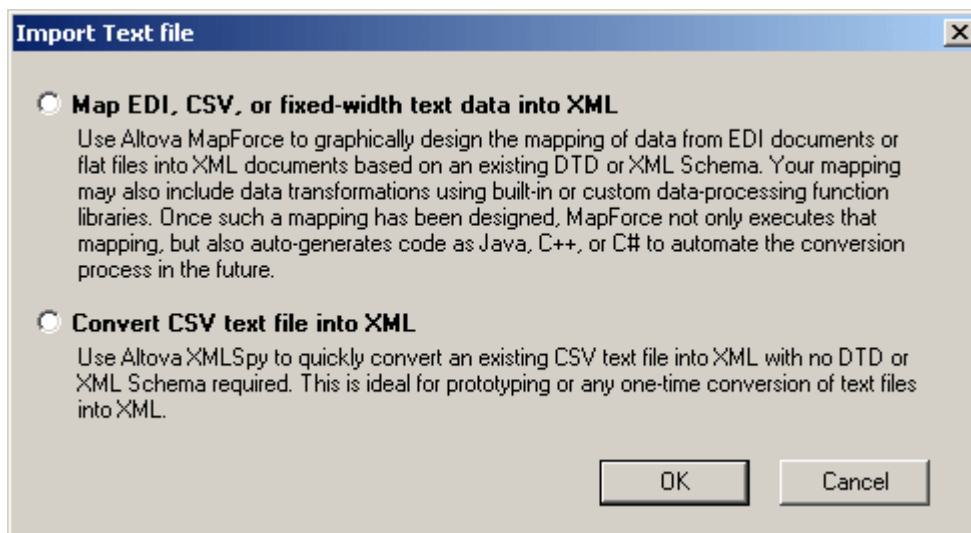


21.10.1 Import Text File

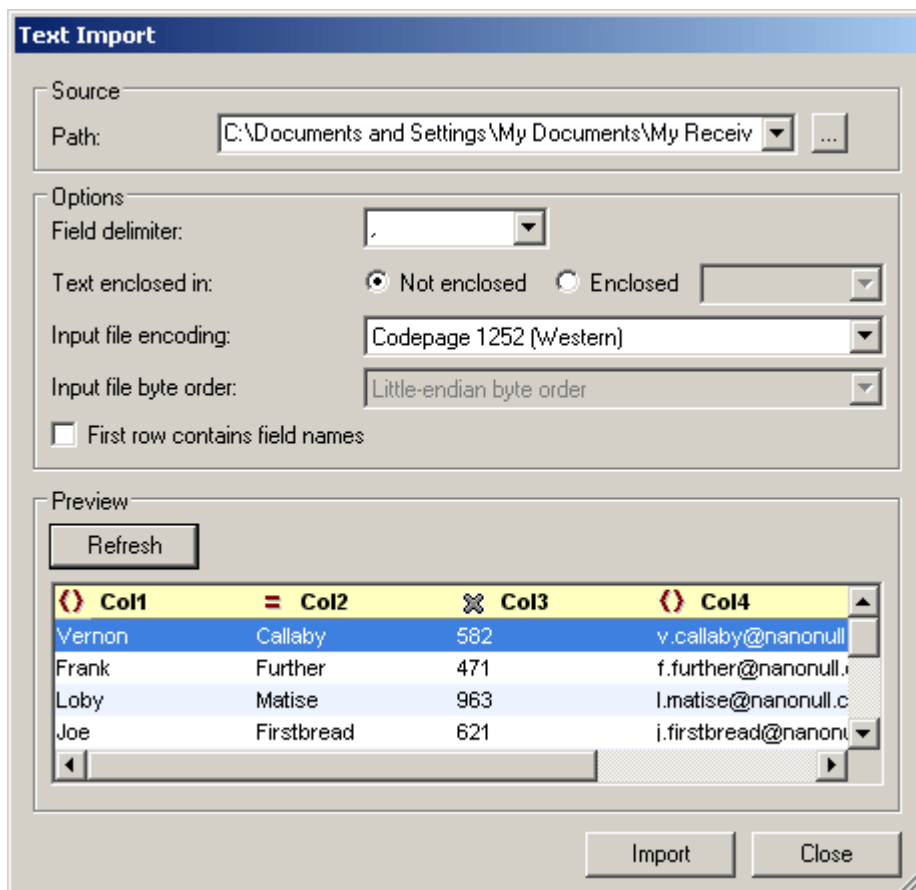


This command lets you **import** any **structured text** file into XMLSpy and convert it to XML format immediately. This is useful when you want to import legacy data from older systems. The steps for importing data in a text file as an XML document are described below.

1. Select the menu item **Convert | Import Text File**. The following dialog appears:



2. Select one of the following:
 - Map EDI, CSV, or fixed-width text data into XML (you must have installed Altova MapForce in order to select this option)
 - Convert CSV text file into XML
3. Click **OK**. The Text import dialog appears (*screenshot below*). How to use this dialog is described below.



Path

Enter the path to the file to import in the Path text box, or select the file using the Browse button to the right of the text box. After the file is selected, a Grid View preview of the XML file is displayed in the Preview pane. Any change in the options selected in this dialog will be reflected in the preview immediately.

Delimiter

To successfully import a text file, you need to specify the field delimiter that is used to separate columns or fields within the file. XMLSpy will auto-detect common row separators (CR, LF, or CR+LF).

String quotes

Text files exported from legacy systems sometimes enclose textual values in quotes to better distinguish them from numeric values. If this is the case, you can specify what kind of quotes are being used in your file, and remove them automatically when the data is imported.

Encoding

The data is converted into [Unicode](#) (the basis of all XML documents), so you need to specify which character-set the file is currently encoded in. For US or Western European Windows systems this will most likely be Codepage 1252, also referred to as the ANSI encoding.

Byte order

If you are importing 16-bit or 32-bit Unicode (UCS-2, UTF-16, or UCS-4) files, you can also switch between little-endian and big-endian byte order.

First row contains column names

It is also very common for text files to contain the field names in the first row within the file. If this is the case, check this check box.

Preview

In the Preview pane you can rename column headers by clicking in a name and editing it. The column headers will be the element or attribute names in the XML document. You can also select whether a column should be an element or an attribute in the XML document, or whether it should not be imported into the XML document. Click the column-type icon in each column header to toggle through these options. In the screenshot above, `Col1` is an element, `Col2` is an attribute, and `Col3` will not be imported.

When you are satisfied with the options, click **Import**. The imported data is converted into an XML document that is displayed in Grid View.

21.10.2 Import Database Data

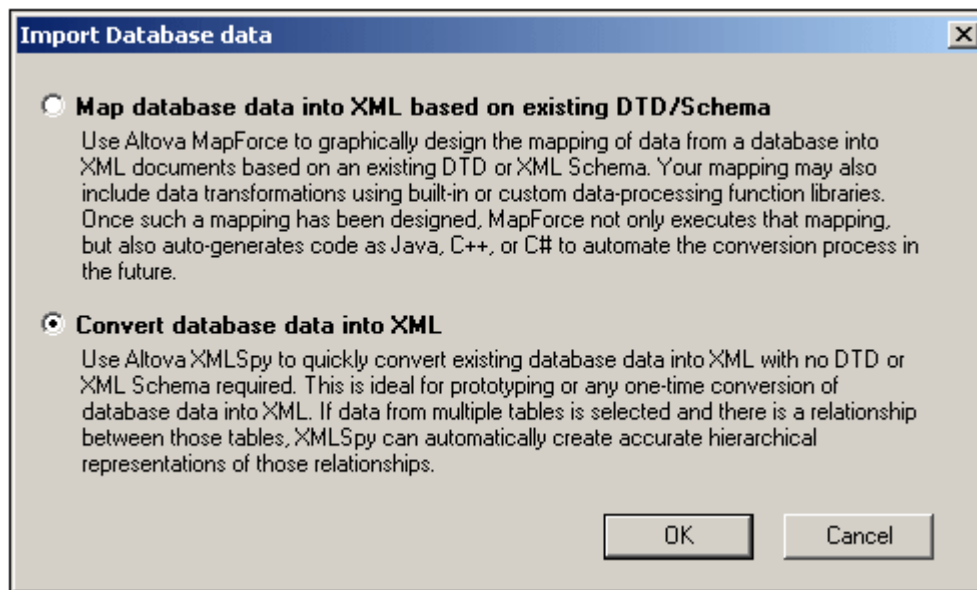


The **Import Database Data** command enables you to import data from any of a variety of databases into an XML file. The import mechanism involves two steps:

1. A [connection to the database is established](#).
2. The [data to be imported is selected](#).

To import database data, do the following:

1. When you click the Import Database Data command, the Import Database Data dialog (screenshot below) pops up.

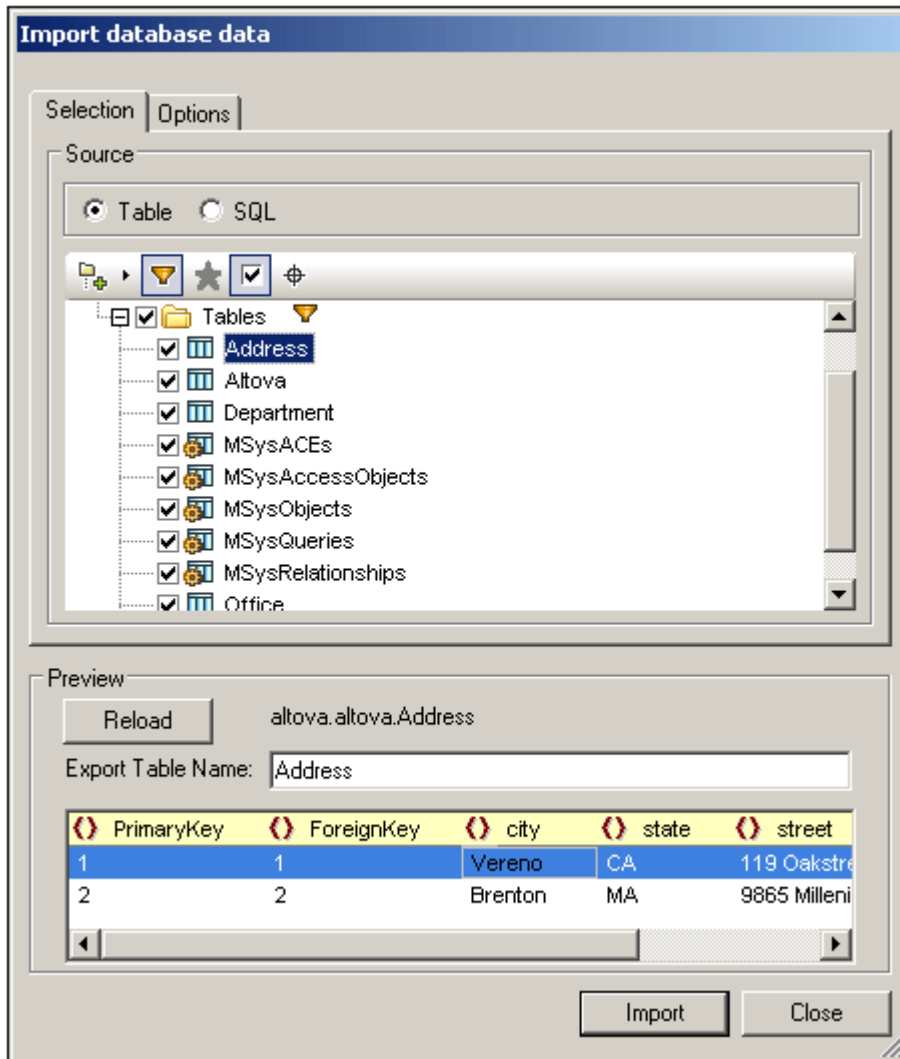


2. Select Convert Database Data into XML and click **OK**. The Connect to a Data Source dialog appears. (The Map Database Data into XML Based on Existing DTD/Schema option requires the use of Altova MapForce to carry out the mapping.)
3. In the [Connect to Data Source](#) dialog, you establish a connection to the database. How to do this for different types of databases is explained in the section, [Connecting to a Data Source](#).
4. After the connection to the database is established, the Import Database Data dialog displays tabs and windows that enable you to select the database data to import. These options are described below. After finishing, click the **Import** button to import the database data.

Data selection and import options

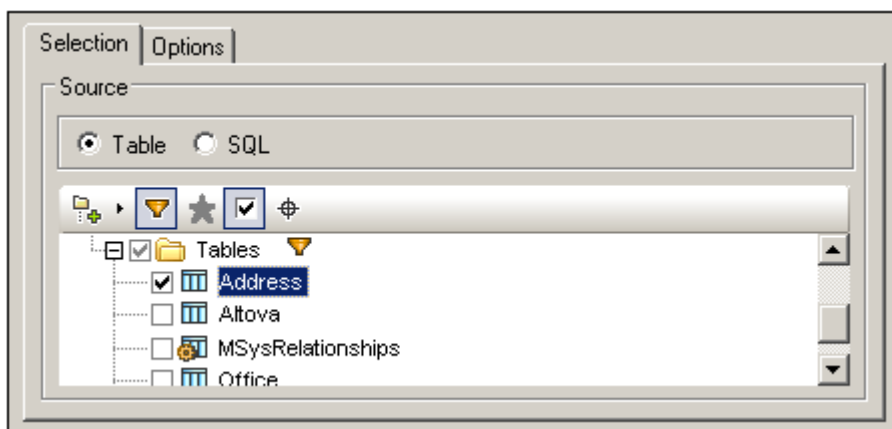
The Import Database Data dialog for setting the selection and import options consists of two parts (*shown separately in the screenshots below*):

- an upper part with two tabs: (i) Selection, and (ii) Options.
- a lower part, which is a Preview window showing the data according to the data selection and import options.



Selection tab

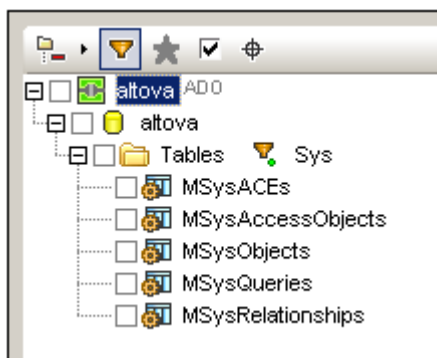
In the Selection tab (*screenshot above*), the Source pane (*screenshot below*) displays either a representation of the tables of the database or an editable SQL statement for selecting the required tables, each view being selected by clicking the respective radio button.



In the Table view, you can select the tables in the database that you want to import by checking the table's check box (*screenshot above*). The contents of the table can then be displayed in the Preview pane. The table selection can be further filtered in the Preview pane (*see below*).

The database structure can be displayed differently and can be filtered. The Layout icon  in the Source pane enables you to organize database objects into: (i) folders based on object type; (ii) folders based on object type, but without schema folders; (iii) in a hierarchy, but without folders; and (iv) categories of tables, based on their relationships with other tables.

Clicking the Filter Folder Contents  icon applies a filter to the selected folder (*in the screenshot below, to the Tables folder*). Clicking the filter icon on the table pops out a menu with a list of filter possibilities. In the screenshot below, the filter has been set to display objects that contain the text `sys` in its name. The view is filtered accordingly.



Clicking the Object Locator icon  pops up a text field, which behaves like a Search entry field. You can enter a text string and the dropdown list will display all the objects whose names contain that text string. Selecting one of these objects from the dropdown list will highlight that object in the tree.

Options tab

In the Options tab (*screenshot below*), you can specify how number, date, and time values are to be imported; whether data is imported as elements or attributes; and whether comments and NULL fields are to be included in the import.

When `NULL` fields are enabled for import, you can enter a substitution XML value for them.

Preview pane

The Preview pane (*screenshot below*) displays the structure of the table currently selected in the Selection tab. When a new table is selected in the Selection tab, click the Preview button in the Preview pane to display the table. Click the Refresh button to refresh the preview.

	Primary	Foreign	city	state	street	zip
1	1		Vereno	CA	119 Oakstreet, Suite 4876	29213
2	2		Brenton	MA	9865 Millenium Center, Suite 4	5985

A field can be specified to be imported as an element or attribute, or not to be imported, by clicking the symbol to the left of the column name. You can click through the element, attribute, and ignore options. In the screenshot above, the `city` field, for example, has been set to be imported as an element while the `street` field has been set to be imported as an attribute.

Datatype conversions

Information about the conversion of database datatypes to XML Schema datatypes is listed in the [Appendices](#).

21.10.3 Import Microsoft Word Document



This command enables the direct **import** of any **Word document** and conversion into XML

format if you have been using paragraph styles in Microsoft Word. This option requires Microsoft Word or Microsoft Office (Version 97 or 2000).

When you select this command, the Open dialog box appears. Select the Word document you want to import.

XMLSpy automatically generates an XML document with included CSS stylesheet. Each Word paragraph generates an XML element, whose name is defined as the name of the corresponding paragraph style in Microsoft Word.

21.10.4 Create XML Schema from DB Structure

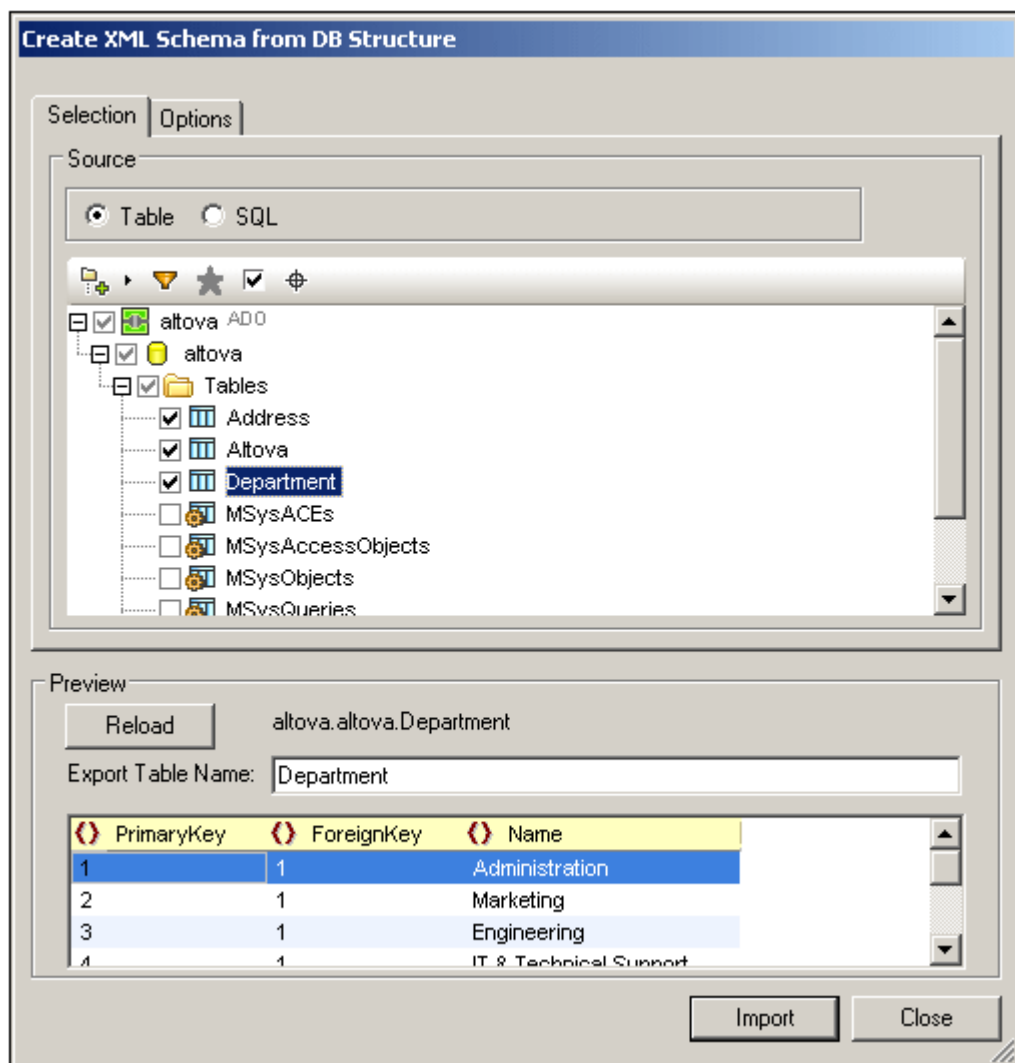


The **Create XML Schema from DB Structure** command enables you to create an XML Schema from the structure of any of a variety of databases. The XML Schema-creation mechanism involves two steps:

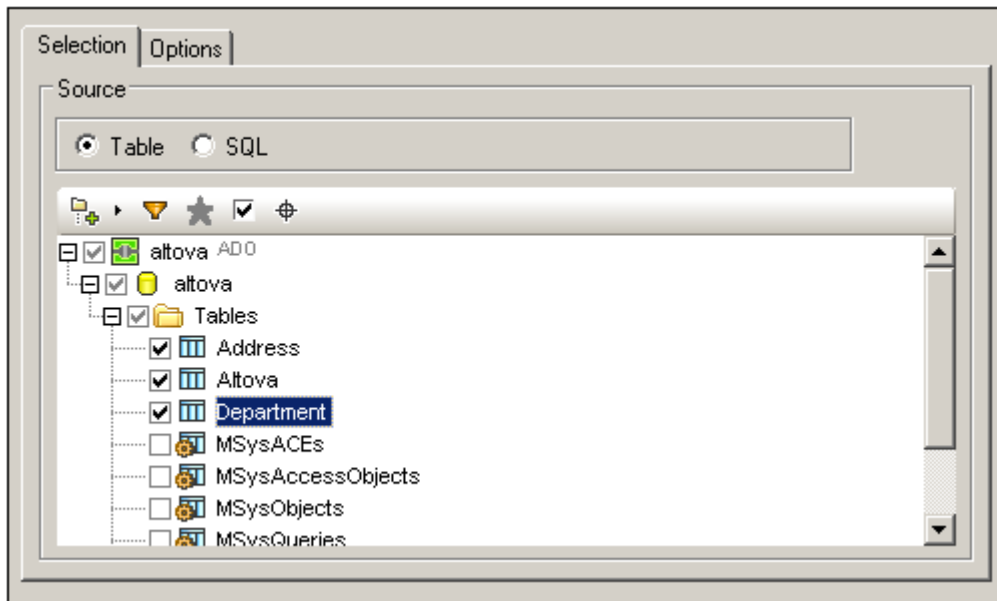
1. A connection to the database is established.
2. Options for the database data selection and the XML Schema are specified.

To create an XML Schema from a DB structure you would do the following:

1. When you click the **Create XML Schema from DB Structure** command, the [Connect to Data Source](#) dialog appears.
2. In the [Connect to Data Source](#) dialog, you establish a connection to the database. How to do this for different types of databases is explained in the section, [Connecting to a Data Source](#).
3. After the connection to the database is established, the Create XML Schema from DB Structure dialog (*screenshot below*) displays tabs and windows that enable you to select the database structure to import. These options are described below. After finishing, click the **Import** button to import the database data.

**Selection tab**

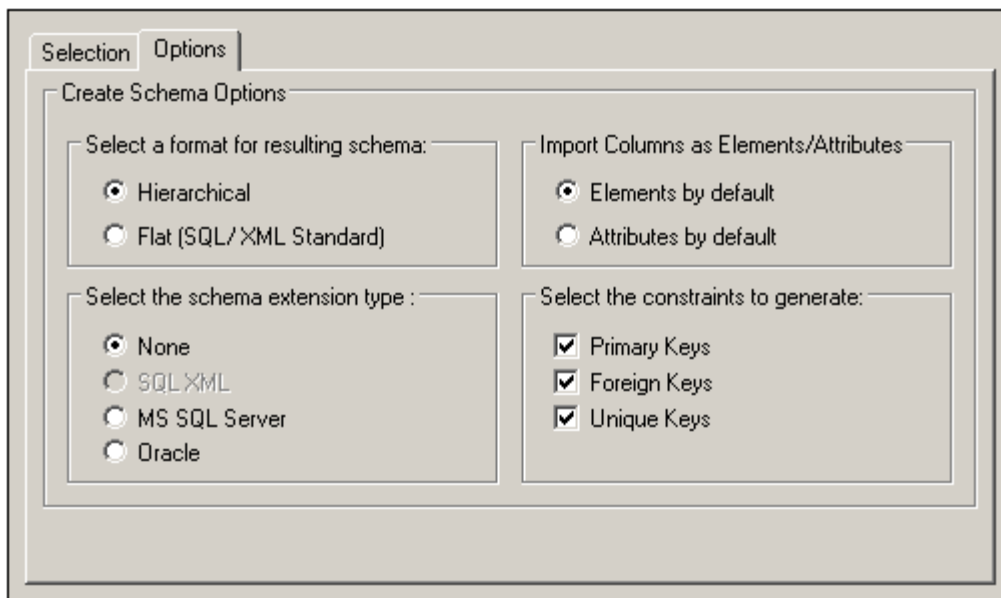
In the Selection tab (*screenshot below*), the data source is listed in the Source Database pane. The Source pane displays either a representation of the tables of the database or an editable SQL statement for selecting the required tables.



In the Table view, you can select the tables in the database that you want to import by checking the table's check box (*screenshot above*). The contents of the table can then be displayed in the Preview pane. The table selection can be further filtered in the Preview pane (*see below*). You can configure the display in the Source pane as described in the section [Import Database Data](#).

Options tab

In the Options tab (*screenshot below*), you can specify the format of the schema, its extension type, whether columns should be imported as elements or attributes, and the database constraints that should be generated in the schema.



Schema format: You can select between a flat (SQL/XML Standard) and a hierarchical schema form

- The **flat** schema model is based on an ISO-ANSI Working draft titled XML-Related

Specification (SQL/XML) (Henceforth referenced as the "SQL/XML Working Draft"). The SQL/XML Working Draft defines how to map databases to XML. Relationships are defined in schema using identity constraints and there are no references to elements. Hence the schema is flat structure which resembles a tree-like view of the database. For more information on the SQL/XML working draft please see:

<http://www.sqlx.org/SQLXdocs>

- The **hierarchical** schema model displays the table dependencies visually, in a type of tree view where dependent tables are shown as indented child elements in the content model. Table dependencies are also displayed in the Identity constraints tab. Tables are listed as global elements in the schema, and columns are the elements or attributes of these global elements (The user decides whether to map the columns as elements or as attributes). Relationships are created in a hierarchical way so that a foreign key field in one table is actually a reference to the global element that represents that table.

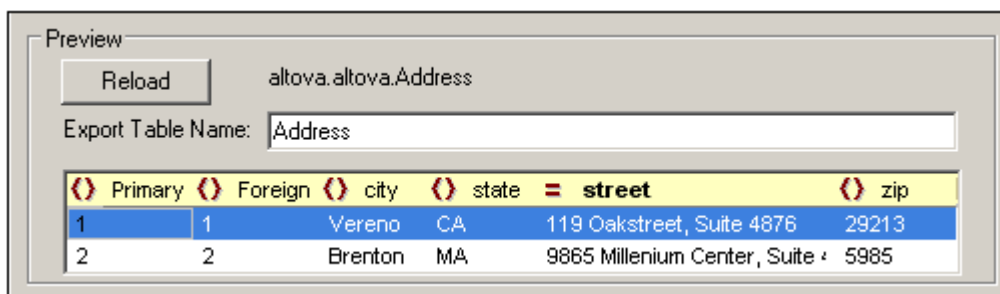
Schema extension type: Schema extension information is additional information read from a database that is then embedded in the schema as either annotation data or attributes. There are four extension type options when generating schemas: (i) no extensions information; (ii) SQL/XML extensions; (iii) MS SQL Server extensions; and (iv) Oracle extensions. These are described below:

- **None:** No additional information is provided by the database.
- **SQL XML:** SQL/XML extensions are only inserted when generating schemas in a Flat Format. The extension information is stored in annotations, and is described in the SQL/XML Working Draft.
- **MS SQL Server:** Selecting Microsoft SQL Server, generates SQL Server extensions. See *Using Annotations in XSL Schemas* in the SQL Server Books Online for detailed information (Download from <http://www.microsoft.com/sql/techinfo/productdoc/2000/books.asp>). The following subset of annotation data are generated in the schema: `sql: relation`, `sql: field`, `sql: datatype`, `sql: mapped`.
- **Oracle:** Oracle extensions are selected by default when working with an Oracle database. Additional database information is stored as attributes. A full description of these attributes can be found at http://download-west.oracle.com/docs/cd/B10501_01/appdev.920/a96620/xd05obj.htm#1027227. The following subset of attributes is currently generated: `SQLName`, `SQLType`, `SQLSchema`.

Note: Although SQL Server and Oracle extensions can be generated for their respective databases they are not restricted in this way. This proves useful when working with a third database and wanting to generate a schema that later should be working with either SQL Server or Oracle.

Preview pane

The Preview pane (*screenshot below*) displays the structure of the table currently selected in the Selection tab. When a new table is selected in the Selection tab, click the Refresh button in the Preview pane to refresh the preview.



A field can be specified to be imported as an element or attribute, or not to be imported, by clicking the symbol to the left of the column name. You can click through the element, attribute, and ignore options. In the screenshot above, the `city` field, for example, has been set to be imported as an element while the `street` field has been set to be imported as an attribute.

Datatype conversions

Information about the conversion of database datatypes to XML Schema datatypes is listed in the [Appendices](#).

21.10.5 DB Import Based on XML Schema



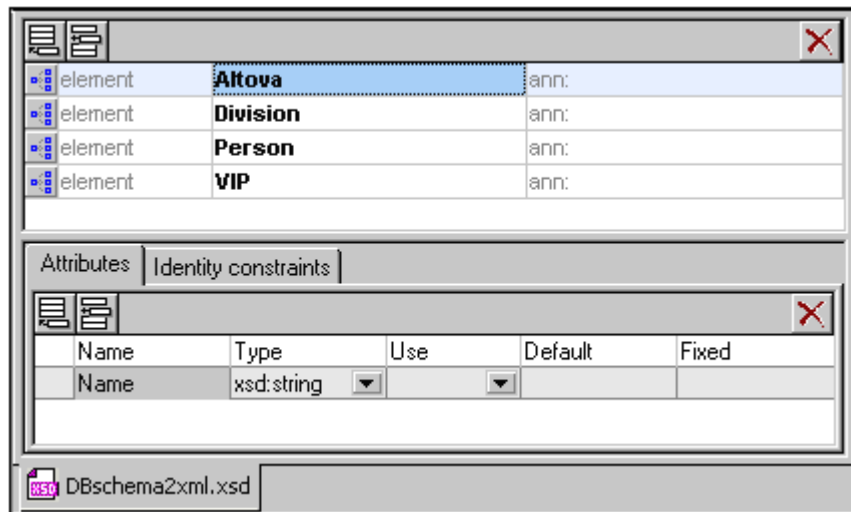
The **DB Import Based on XML Schema** command creates an XML document which is valid according to a given XML Schema and contains data imported from a database. For this feature, the following databases are supported:

- Microsoft Access 2000 and 2003
- Microsoft SQL Server
- Oracle
- MySQL
- Sybase
- IBM DB2

The data to be imported is determined by the table that is selected in the database. With the required XML Schema (that on which you wish to base the import) as the active document in Schema View, connect to the database. Then select the table/s you wish to import, and click Import. The data is imported into an XML document, and the document has the structure of the XML Schema that was active when the data was imported.

In the example below, data from an MS Access database is imported with an XML Schema active in Schema View. These would be the steps to carry out for the import:

1. Open the schema file in Schema View (*screenshot below*).



2. Select the menu command **DB Import based on XML Schema**. This opens the [Connect to Data Source](#) dialog.
3. Select the Microsoft Access (ADO) option and click **Next**.
4. Click **Browse** and select the database file. Then click **Next**.
5. In the DB Import Based on XML Schema dialog which pops up, go to the Tables tab, select one or more tables you wish to import (for example, *Altova*), then click **Import**. The table is imported into an XML document that is displayed in Grid View.

Datatype conversions

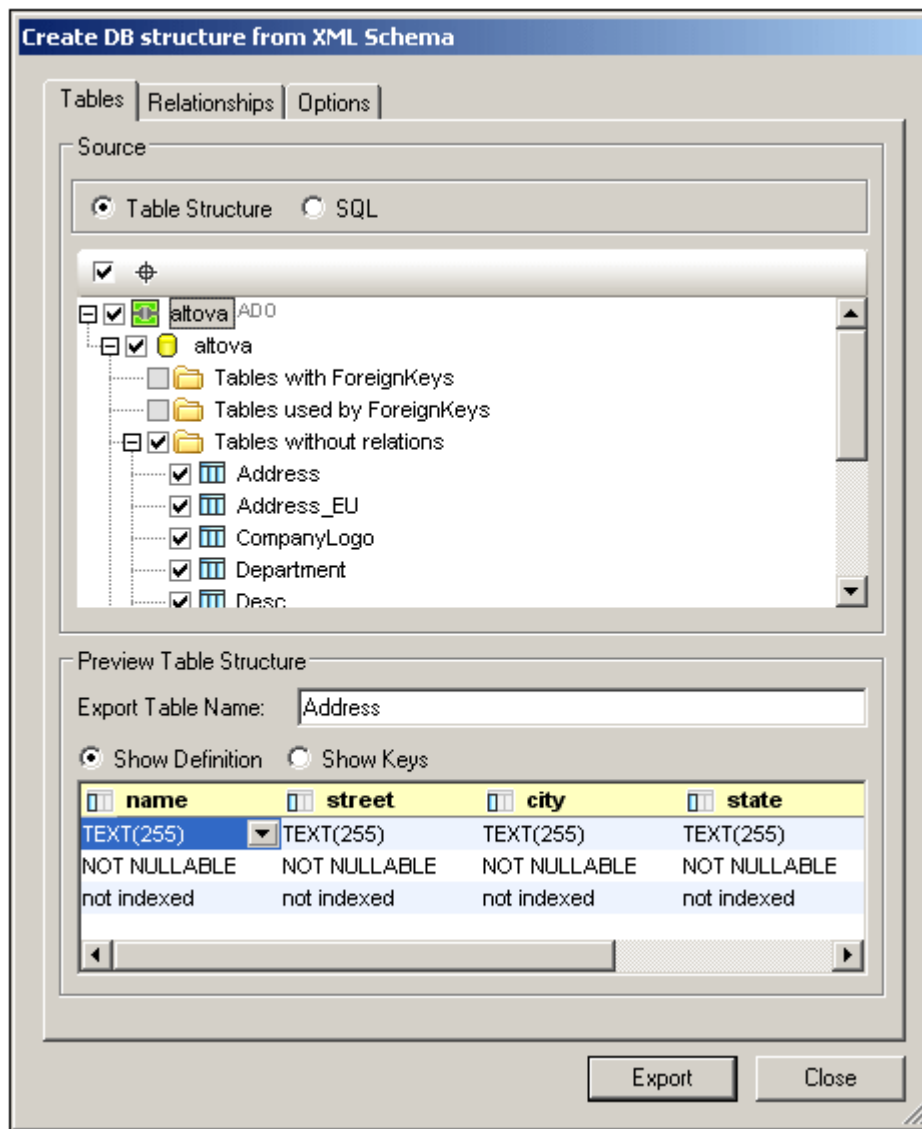
Information about the conversion of database datatypes to XML Schema datatypes is listed in the [Appendices](#).

21.10.6 Create DB Structure from XML Schema



XMLSpy allows you to create an empty database (or skeleton database) based on an existing schema file. The method described below is generally the same for each type of database.

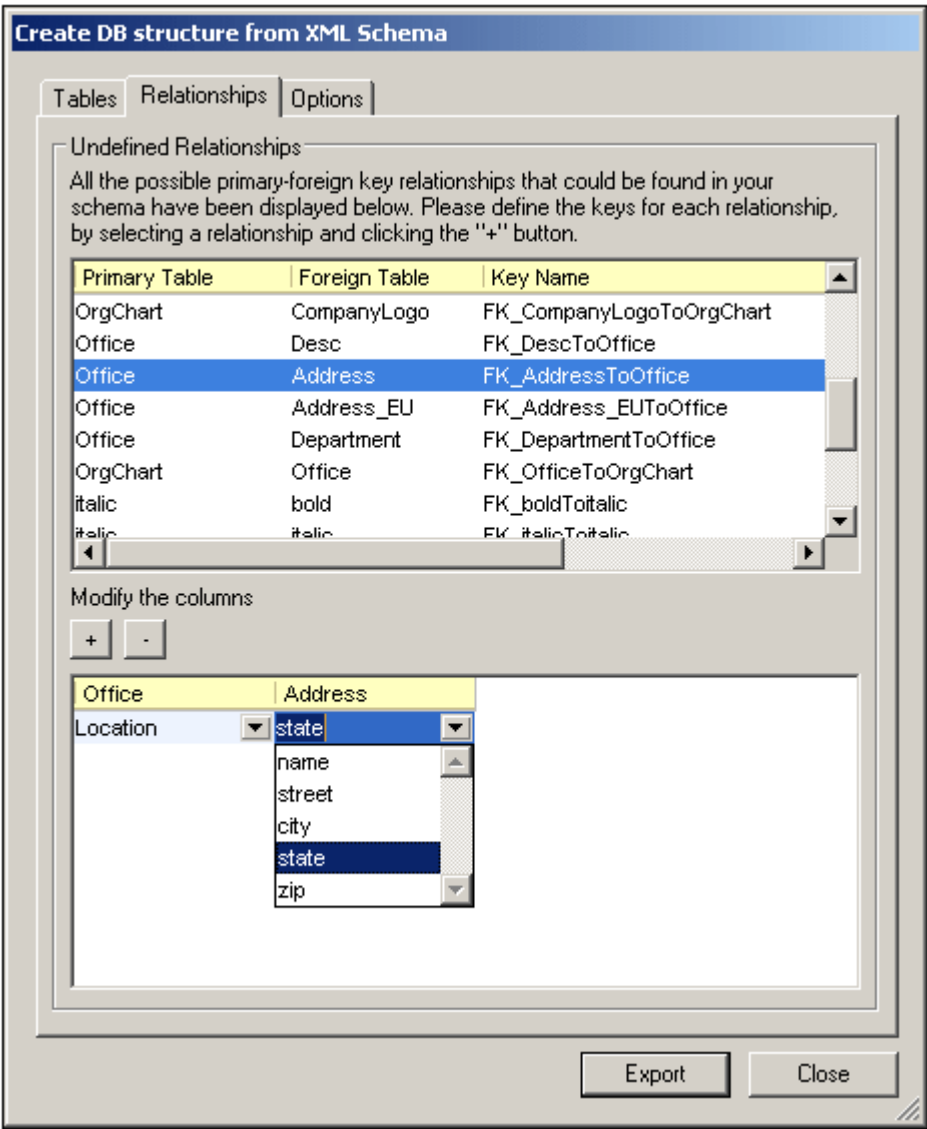
1. Open the schema file in Schema/WSDL View
2. Select the menu command **Convert | Create DB Structure from XML Schema**. This pops up the [Connect to a Data Source](#) dialog, which enables you to connect to a database (DB).
3. Use the steps described in the section [Connecting to a Data Source](#) to connect to the required database. For example, to connect to a Microsoft Access database, select the Microsoft Access radio button, and continue the process to select a database. You can use an existing database or create a new database in which the schema structure will be contained.
4. In the Create DB Structure from XML Schema dialog, tables are created from the schema and displayed in a tree format at the location where they will occur in the DB. For example, in the screenshot below, the Address table is created and selected for export. Tables that should not be exported should be deselected (by unchecking the check box or selecting the appropriate item from the context menu for that table).



Creating DB tables with relationships

If the XML Schema from which the DB structure is generated has relationships defined in the form of identity constraints, then these relationships are automatically created in the generated DB structure and displayed in the Table Structure. Tables with relationships are listed under the sections: Tables with ForeignKeys and Tables used by ForeignKeys. Tables without relationships are listed in the Independent Tables section.

In the Relationships tab, you can create and modify table relationships. The tab lists all possible primary-key/foreign-key relationships (*screenshot below*).



To create a relationship, do the following:

1. Select one of the possible primary-key/foreign-key relationships.
2. In the lower pane of the dialog, click the Plus button to create a relationship.
3. Select the required columns in each of the two tables from the respective dropdown lists.

You can also remove a relationship by selecting it and then clicking the Minus button.

Notes on database structure and connecting

The schema structure, defined by the identity constraints, is mirrored in the resulting database. The table below shows the type of database created, the restrictions, and the connecting methods, when using the **Create DB Structure from XML Schema** menu command.

	Directly	using ODBC	using ADO
MS Access (2000 and 2003)	OK *	OK	OK

MS SQL Server	OK *	OK	OK
Oracle	OK *	OK	OK
MySQL	-	OK *	OK †
Sybase	-	OK *	OK
IBM DB2	-	OK *	OK

* *Recommended connection method for each database.*

† *MySQL: When creating the ADO connection based on ODBC, it is recommended to use either the User or System DSN.*

- *Not supported*

XMLSpy will map both [hierarchical and flat formatted schemas](#). XMLSpy recognizes both formats automatically.

The flat format is mapped to SQL in two different ways.

- SQL Server DB, Oracle DB, or Sybase DB:
A schema that was generated in flat format, for one of the above databases, will have the schema catalog name extracted and used in the generated SQL script as the DB name. This means that the resulting SQL script will be executed on a target DB whose name must be identical to the schema catalog name.
- Access (2000 or 2003), MySQL, or DB2 DB:
A schema that was generated in flat format, for one of the above databases, will **ignore** the schema catalog name when the SQL script is generated. This means that the resulting SQL script will be executed on a target database that was logged into.

Datatype conversions

Information about the conversion of XML Schema datatypes to database datatypes is listed in the [Appendices](#).

21.10.7 Export to Text Files



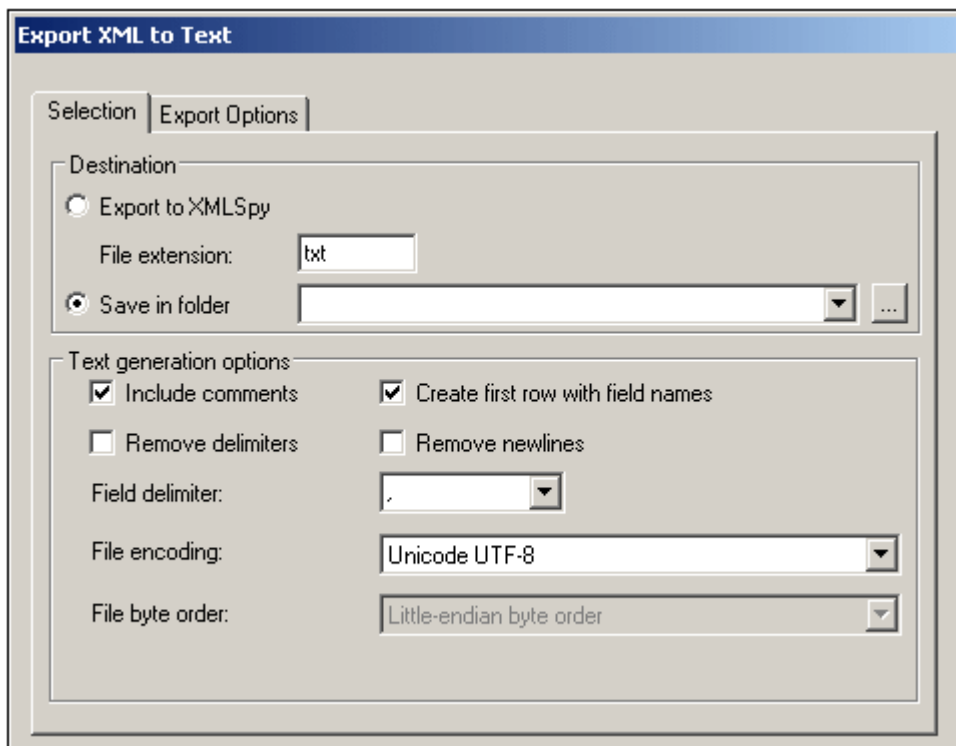
The command **Convert | Export to Text Files** exports XML data into text formats for exchange with databases or legacy systems. On clicking this command, the Export XML to Text dialog pops. It consists of two parts (*shown separately in the screenshots below*):

- an upper part with two tabs: (i) Selection, and (ii) Export Options.
- a lower part, which is a Preview window.

After you have selected the desired options in this dialog (*described below*), click the **Export** button to export to text file/s.

Selection

In the Selection tab (*screenshot below*), you can select the destination of the file to be exported and text generation options.



Destination: The exported file can be saved directly to a folder. The file extension can be specified. The filenames will be those of the elements (in the XML file) that will be exported. Alternatively, untitled files can be exported to XMLSpy. These files will be displayed in the GUI, and can be saved later.

Include comments: Activate this option to include a comment in the exported XML file that shows the SQL query used to select the data, as well as a list containing one item for each column header in the database table.

Create first row with field names: When activated, the exported tables include the database column names.

Remove delimiters: Removes delimiters that are contained in text values in the exported data. Set the delimiter you want to remove by using the Delimiter combo box in this tab. For example, if this option is activated and the selected delimiter is the apostrophe, when you export the XML value `Ba'ker`, the string will be `Baker` in the exported text.

Remove newlines: Removes newlines from exported data.

Delimiter: Select from the drop-down list the character that you wish to have removed during export. Alternatively, enter the desired character string.

Encoding: Select from the drop-down list, the desired encoding for files that are generated during export.

Byte order: If you are exporting 16-bit or 32-bit Unicode (UCS-2, UTF-16, or UCS-4) files, you can also switch between little-endian and big-endian byte order.

Export Options

Additional export options, which are described below, can be specified in the Export Options tab (*screenshot below*):

The screenshot shows the 'Export Options' dialog box. It has two tabs: 'Selection' and 'Export Options'. The 'Export Options' tab is selected. The dialog is divided into several sections. The 'Start point of export' section has two radio buttons: 'Starting from XML-file begin' (selected) and 'Starting from' followed by a dropdown menu. The 'Export depth' section has two radio buttons: 'Export all sub-elements' (selected) and 'Limited to:' followed by a text box containing '0' and the label 'sub-levels'. The 'Export fields' section has four checkboxes: 'Create from attributes' (checked), 'Create from text values' (checked), 'Apply text value to parent' (unchecked), and 'Convert entities to text' (checked). The 'Automatic fields' section has two checkboxes: 'Create primary/foreign keys' (checked) and 'Independent primary key counter for every element' (checked). The 'Other Options' section has two radio buttons: 'Exclude namespace name' (selected) and 'Replace colon with underscore' (unchecked). An 'Apply Options' button is located at the bottom left of the dialog.

Start point of export: You can choose to export the entire XML document or restrict your export to the data hierarchy starting from the currently selected element. The number of sub-levels below the start point that will be exported is specified in the Export Depth option.

Export depth: Specifies the number of sub-levels below the start point that will be exported.

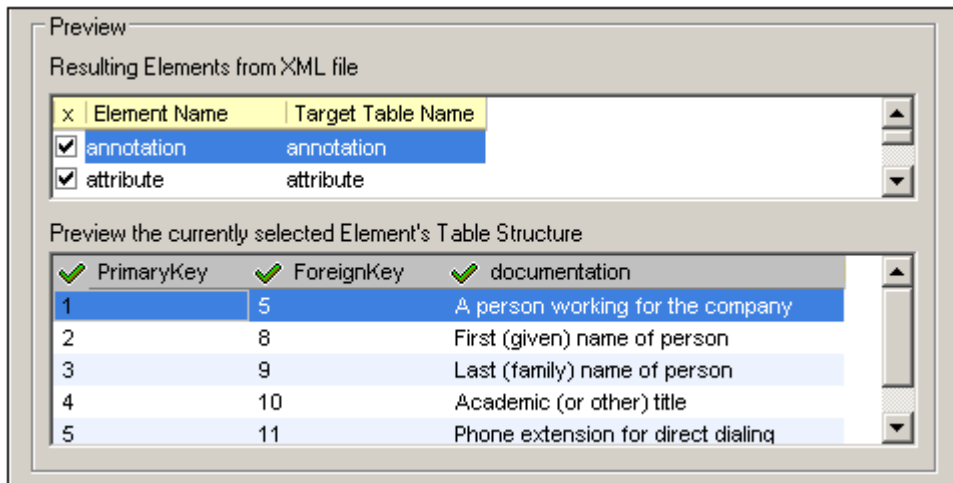
Export fields: Depending on your XML data, you may want to export only elements, attributes, or the textual content of your elements. Note that you can deselect the export of individual elements in the Preview window.

Automatic fields: XMLSpy will produce one output file or table for each element type selected. You can choose to automatically create primary/foreign key pairs to link your data in the relational model, or define a primary key for each element.

Exclude namespace name: Together with the Replace Colon With Underscore radio button this is an either/or choice. Specifies whether namespace prefixes of elements and attributes should be excluded or whether the colon in the namespace prefix should be replaced with an underscore.

Preview window

The Preview window (*screenshot below*) is displayed below the Selection and Export Options tab.



The Resulting Elements from XML File pane shows the node name that will be exported and the name in the generated file. You can select/deselect nodes that will be exported. When an element is selected, a preview of its structure is shown in a second pane below. In this pane, clicking to the left of a column name toggles the export of that column on and off.

21.10.8 Export to a Database



The command **Convert | Export to a Database** exports XML data to a database. On clicking this command, the Connection Wizard starts up and enables you to set up a connection to the database you wish to update. After a connection has been established, the Export Data to Database dialog pops. It consists of two parts (*shown separately in the screenshots below*):

- an upper part with two tabs: (i) Selection, and (ii) Export Options.
- a lower part, which is a Preview window.

After you have selected the desired options in this dialog (*described below*), click the **Export** button to export to the database.

Selection

In the Selection tab, you can select the destination database and table generation options. The destination field selects the connection to the database. You must select whether the data is created as new tables, updates existing tables, or first tries to update an existing table and then creates a new table if an update is not possible. You can also set a stop action based on the number of errors, and, optionally, SQL script logging.

Export Options

Export options, which are described below, can be specified in the Export Options tab (*screenshot below*):

The screenshot shows the 'Export Options' dialog box. It has two tabs: 'Selection' and 'Export Options'. The 'Export Options' tab is selected. The dialog is organized into several sections. The 'Start point of export' section has two radio buttons: 'Starting from XML-file begin' (selected) and 'Starting from' followed by a dropdown menu. The 'Export depth' section has two radio buttons: 'Export all sub-elements' (selected) and 'Limited to: 0 sub-levels'. The 'Export fields' section has four checkboxes: 'Create from attributes' (checked), 'Create from text values' (checked), 'Apply text value to parent' (unchecked), and 'Convert entities to text' (checked). The 'Automatic fields' section has two checkboxes: 'Create primary/foreign keys' (checked) and 'Independent primary key counter for every element' (checked). The 'Other Options' section has two radio buttons: 'Exclude namespace name' (selected) and 'Replace colon with underscore' (unchecked). An 'Apply Options' button is located at the bottom left of the dialog.

Start point of export: You can choose to export the entire XML document or restrict your export to the data hierarchy starting from the currently selected element. The number of sub-levels below the start point that will be exported is specified in the Export Depth option.

Export depth: Specifies the number of sub-levels below the start point that will be exported.

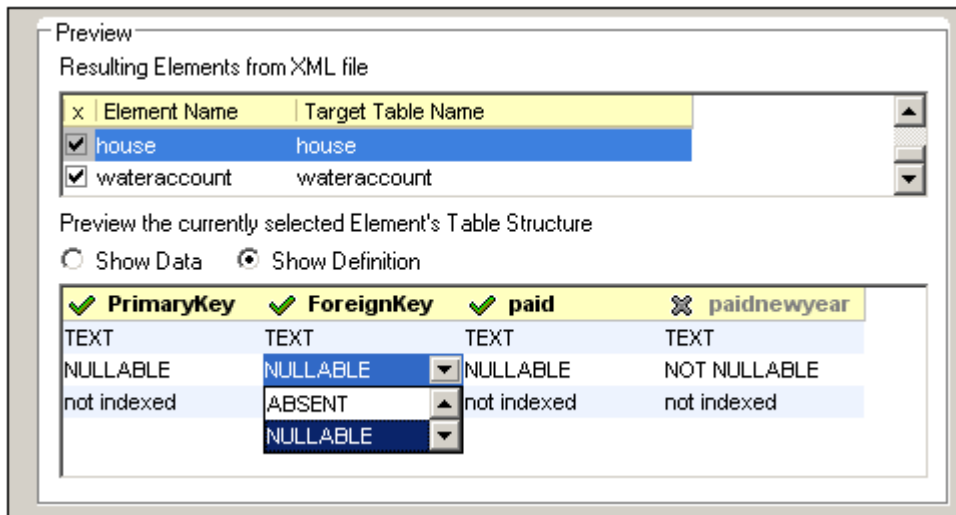
Export fields: Depending on your XML data, you may want to export only elements, attributes, or the textual content of your elements. Note that you can deselect the export of individual elements in the Preview window.

Automatic fields: XMLSpy will produce one output file or table for each element type selected. You can choose to automatically create primary/foreign key pairs to link your data in the relational model, or define a primary key for each element.

Exclude namespace name: Together with the Replace Colon With Underscore radio button this is an either/or choice. Specifies whether namespace prefixes of elements and attributes should be excluded or whether the colon in the namespace prefix should be replaced with an underscore.

Preview window

The Preview window (*screenshot below*) is displayed below the Selection and Export Options tab.



The Resulting Elements from XML File pane, shows the node name that will be exported and the name in the generated file. You can select/deselect nodes that will be exported. When an element is selected, a preview of its structure is shown in a second pane below. In this pane, clicking to the left of a column name toggles the export of that column on and off. In the screenshot above, the `paidnewyear` column has been toggled off, so it will not be exported.

When the element's table structure shows the data, the key constraint type can be set by clicking to the left of a column name. When the element's table structure shows field definitions, the definitions can be edited by selecting the definition and selecting an option from the definition's combo box (see screenshot above).

21.10.9 Convert XML to/from JSON

The **Convert XML to/from JSON** command converts the active document, if XML, to a JSON document. If the active document is a JSON document, clicking the **Convert XML to/from JSON** command will convert the JSON document to an XML document. The generated document can then be saved to any location.

JSON (JavaScript Object Notation) is a data interchange format based on a subset of JavaScript and is commonly used with JavaScript code. Code for parsing JSON is widely available for many programming languages.

XMLSpy provides intelligent editing features for JSON documents. These are described in the [JSON Editing Features](#) section of the documentation.

Given below is an example of a source XML document, and, below it, the JSON document generated by the **Convert XML to/from JSON** command.

XML document

```
<?xml version="1.0" encoding="UTF-8"?>
<Person first="Jim" last="James">
  <Address>
    <street>4 New Street</street>
    <city>New York</city>
    <state>NY</state>
    <code>10123</code>
  </Address>
  <Tel type="home">
    123 123-1234
  </Tel>
  <Tel type="office">
```

```

123 987-9876
</Tel>
</Person>

```

JSON document

```

{
  "XML": {
    "version": 1.0,
    "encoding": "UTF-8"
  },
  "Person": {
    "first": "Jim",
    "last": "James",
    "Address": {
      "street": "4 New Street",
      "city": "New York",
      "state": "NY",
      "code": 10123
    },
    "Tel": [ { "type": "home",
      "Text": "\r      123 123-1234\r      " }, { "type": "office",
      "Text": "\r      123 987-9876\r      " } ]
  }
}

```

To convert a JSON document to XML, make the JSON document active and click the **Convert XML to/from JSON** command.

21.11 View Menu

The **View** menu controls the display of the active [Main window](#) and allows you to change the way XMLSpy displays your XML documents.

This section provides a complete description of commands in the **View** menu.

21.11.1 Text View



This command **switches** the current view of the document to [Text View](#), which enables you to edit the document in its text form. It supports a number of advanced text editing features, described in detail in [Text View](#) section of this document.

Please note: You can configure aspects of the Text View using options available in the various tabs of the Options dialog ([Tools | Options](#)).

21.11.2 Enhanced Grid View



This command **switches** the current document into [Grid View](#).

If the previous view was the [Text View](#), the document is automatically checked for well-formedness.

ipo:purchaseOrder	
xmlns:xsi	http://www.w3.org/2001/XMLSchema-instance
xmlns:ipo	http://www.altova.com/IPO
orderDate	1999-12-01
xsi:schema...	http://www.altova.com/IPO ipo.xsd
shipTo export-code=1 xsi:type=ipo:EU-Address	
billTo	
xsi:type	ipo:US-Address
name	Robert Smith
street	8 Oak Avenue
city	Old Town
state	AK
zip	95819

Embedded Database/Table view

XMLSpy allows you to display recurring elements in a [Database/Table View](#) from within the Grid View. This function is available wherever the Grid View can be activated, and can be used when editing any type of XML file - XML, XSD, XSL etc.

For further information on this view, please see the [Grid View](#) section of this documentation.

21.11.3 Schema Design View



This command **switches** the current document to Schema Design View. The switch will be successful only if the document is an XML Schema document. This view is described in detail in the [Schema View](#) section of this documentation.

21.11.4 WSDL Design View

This command **switches** the current document to WSDL Design View. The switch will be successful only if the document is a WSDL document. This view is described in detail in the [WSDL View](#) section of this documentation.

21.11.5 XBRL Taxonomy View

This command **switches** the current document to XBRL Taxonomy View. The switch will be successful only if the document is an XBRL document. This view is described in detail in the [XBRL View](#) section of this documentation.

21.11.6 Authentic View



This command **switches** the current document into the [Authentic View](#).

Authentic View enables you to edit XML documents **based on StyleVision Power Stylesheet templates created in StyleVision!** The templates in StyleVision are saved as **StyleVision Power Stylesheets** (*.sps files), and supply all the necessary information needed by Authentic View.

To **open** a template select the **File | New** command and then click the **Select a StyleVision stylesheet...** button. Please see the [Authentic View](#) documentation for further information.

Please note: If, when you try to switch to Authentic View, you receive a message saying that a temporary (temp) file could not be created, contact your system administrator. The system administrator must change the default Security ID for "non-power users" to allow them to create folders and files.

21.11.7 Browser View



This command **switches** the current document into [Browser View](#).

This view uses an XML-enabled browser to render the XML document using information from potential CSS or XSL style-sheets.

When switching to browser view, the document is first checked for validity, if you have selected Validate upon saving in the [File tab of the Options dialog](#). Use the menu command **Tools | Options** to open this dialog.

For further information on this view, please see the detailed description of the various views in the [Main Window](#) section.

21.11.8 Expand



Hotkey: **"+" on the numeric keypad**

This command **expands** the selected element by one level.

The command can be used in the Grid View.

In the Grid View, the element and all its children remain selected after expansion. This allows you to expand a large element by pressing the **+** key repeatedly.

You can expand and collapse any element by clicking on the gray bar to the left of each element.

21.11.9 Collapse



Hotkey: "-" on the numeric keypad

This command **collapses** the selected element by one level.

The command can be used in the Grid View.

You can expand and collapse any element by clicking on the gray bar to the left of each element.

21.11.10 Expand Fully



Hotkey: "+" on the numeric keypad

This command **expands** all child items of the **selected element**, down to the last level of nesting.

The command can be used in the Grid View.

21.11.11 Collapse Unselected

Hotkey: **CTRL + "-"** key on the numeric keypad

This command allows you to focus on one element and its children, and ignore all the other surrounding elements.

The command can be used in the Grid View.

Select the item that you want to work with and choose this command to collapse all other (unselected) elements.

21.11.12 Optimal Widths



This command adjusts the widths of all columns so that each has a width that exactly accommodates in one line the longest text string in any of its cells. A maximum optimal width can be specified in the View tab of the Options dialog (**Tools | Menu**). Note that optimal widths are calculated on the basis of the visible cells of columns. This enables the optimization of the view when individual elements are collapsed or expanded.

21.11.13 Word Wrap



This command enables or disables word wrapping in the **Text view**.

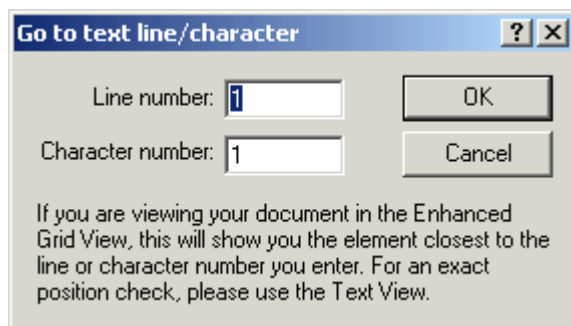
21.11.14 Go to Line/Character



Hotkey: **CTRL+ g**

This command goes to a **specific line number** and/or character position in an XML document in the Text view.

If you are working with an external XSLT processor (see the [XSL tab on the Tools | Options dialog](#) for details) you may often get error messages by line number and character position. XMLSpy lets you quickly navigate to that spot, using this command:



This command works in both the [Text View](#) and [Grid View](#), but will only be able to show an approximate position in the grid view by highlighting the element closest to the character position specified.

21.11.15 Go to File



This command **opens** a document that is being **referred** to, from within the file you are currently editing.

Select the file name, path name, or URL you are interested in, and choose this command from the [View menu](#).

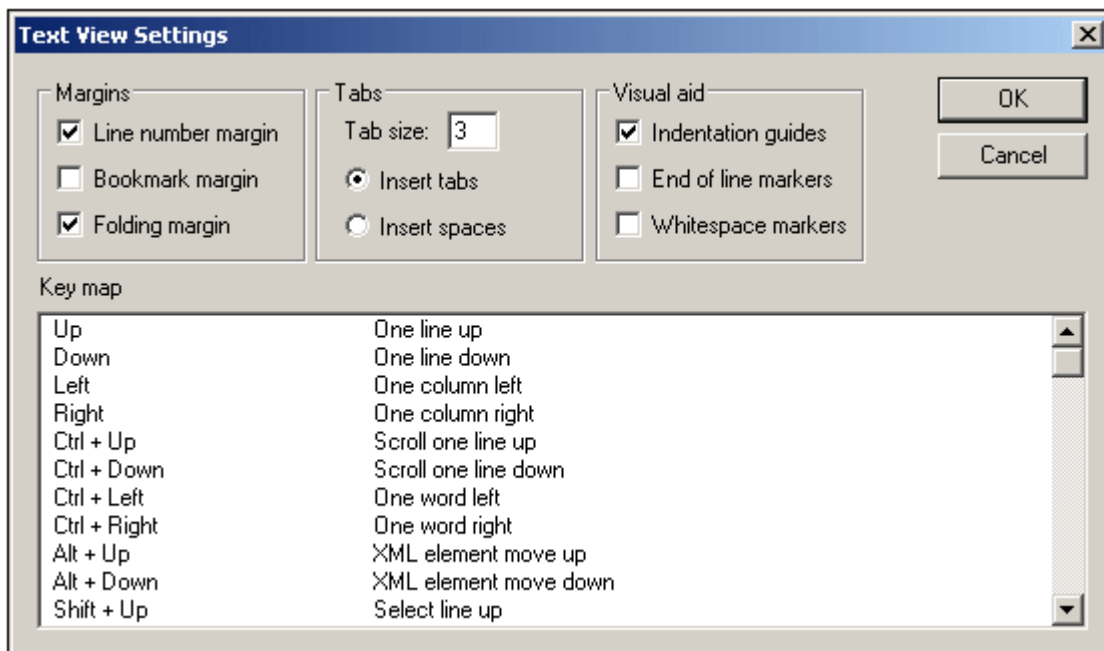
You can select:

- An entire element or attribute in the [Grid View](#)
- Some characters from within any item in the [Text View](#) or [Grid View](#).
- An enclosed string. If your text cursor is between quotes, XMLSpy will automatically use the entire string that is enclosed in the quotes.

21.11.16 Text View Settings



The **Text View Settings** command opens the Text View Settings dialog (*screenshot below*), in which you can configure Text View. The shortcut for the command is available as an icon in the Text toolbar.



Margins

In the Margins pane, the Line Number, Bookmark, and Source Folding margins can be toggled on and off. These are separate margins and display, respectively, line numbers, bookmarks, and source folding (icons to expand and collapse nodes). These settings determine whether the margins are displayed in Text View or not. Bookmark commands are in the **Edit** menu. To expand and collapse nodes, the Folding margin must be toggled on.

Tabs

The Tab pane enables you to set the tab size in terms of spaces. The radio buttons for inserting either tabs or spaces determine whether documents are displayed with tab or space indentation when [pretty-printing with indentation](#) is toggled on.

Visual Aid

The Visual Aid pane contains settings to toggle on indentation guides (dotted vertical lines that show the indentation of the text), end-of-line markers, and whitespace markers.

Key map

The key map is a list of XMLSpy shortcuts and their associated commands.

21.12 Browser Menu

The commands in the **Browser** menu are enabled in [Browser View](#) only.

21.12.1 Back



Backspace (Alt + Left Arrow)

In Browser View, the **Back** command displays the previously viewed page. The Backspace key also achieves the same effect. The **Back** command is useful if you click a link in your XML document and want to return to your XML document.

In Schema View, the **Back** command takes you to the previously viewed component. The shortcut key is **Alt + Left Arrow**. Using the **Back** command up to 500 previously viewed positions can be re-viewed.

21.12.2 Forward



(Alt + Right Arrow)

The **Forward** command is only available once you have used the **Back** command. It moves you forward through (i) previously viewed pages in Browser View, and (ii) previous views of schema components in Schema View.

21.12.3 Stop



The **Stop** command instructs the browser to stop loading your document. This is useful if large external files or graphics are being downloaded over a slow Internet connection, and you wish to stop the process.

21.12.4 Refresh



F5

The **Refresh (F5)** command updates the Browser View by reloading the document and related documents, such as CSS and XSL stylesheets, and DTDs.

21.12.5 Fonts

The **Fonts** command allows you to select the default font size for rendering the text of your XML document. It is similar to the Font Size command in most browsers.

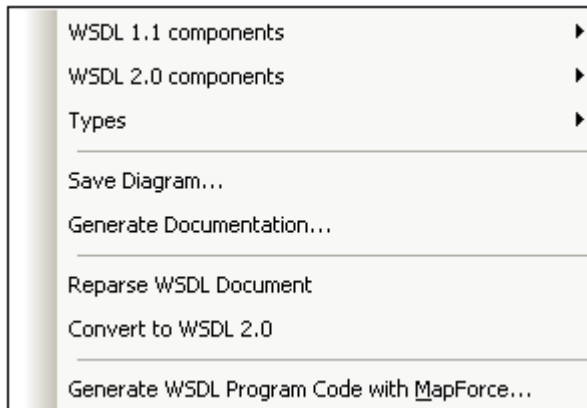
21.12.6 Separate Window



The **Separate Window** command opens the Browser View in a separate window, so that side-by-side viewing is possible. If you have separated the Browser View, press **F5** in editing view to automatically refresh the corresponding Browser View. To dock separate windows back into the interface, click the maximize button (at top right) of the active window.

21.13 WSDL Menu

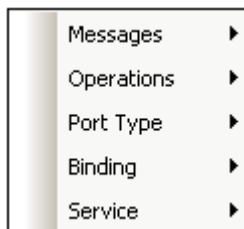
The commands in the **WSDL** menu are available when viewing a WSDL document in [WSDL View](#), which is graphical editor for creating and editing WSDL documents..



For a description of WSDL View, see the section [WSDL View](#). To get started with WSDL, see the [WSDL Tutorial](#).

21.13.1 WSDL 1.1 Components

Mousing over the **WSDL 1.1 Components** menu item scrolls out a submenu (*screenshot below*) from which various commands for editing WSDL 1.1 components can be selected.



Each item of the WSDL 1.1 menu (*screenshot above*) rolls out its own submenu, from which commands relating to that component can be selected. The commands in each of these submenus are described in the subsections of this section.

Messages

Insert message

Adds a new message to the WSDL document. The Messages item in the Overview entry helper opens, and the newly created message is highlighted there.

Delete message

Deletes the selected message from the input or output element.

Add message part (parameter)

Adds a message part (parameter) to the selected message.

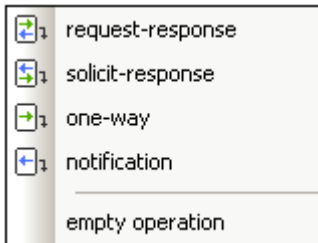
Delete message part (parameter)

Deletes a message part (parameter) from the selected message.

Operations

Append Operation

Appends a new operation to the Operations window. The type of operation to be appended can be selected from the submenu (*screenshot below*) of the **Append Operation** menu command.



Delete Operation

Deletes the selected operation from the Operations window.

Add Input Element

Adds an input element to the selected operation.

Add Output Element

Adds an output element to the selected operation.

Add Fault Element

Adds a Fault element to the selected operation.

Delete Input/Output/Fault Element

Deletes the selected input, output or fault element.

Add New Message to Input/Output/Fault Element

Adds a new (default) message, to the currently selected **input** or **output** message.

PortType

Insert PortType

Adds a new PortType to the PortTypes window.

Delete PortType

Deletes the selected PortType from the PortTypes window.

Binding

Insert Binding

Adds a new binding to the Bindings window.

Delete Binding

Deletes the selected binding from the Bindings window.

Append Child

Enables the addition of a new extensibility element to an input or output message. If the menu item is unavailable, it is not allowed in this position. See the W3C WSDL Specs for more information on Extensibility items.

The following extensibility items are available:

- soap:body
- soap:header
- soap:headerfault
- soap:fault
- mime:content
- mime:multipartrelated
- mime:part
- mime:mimeXml
- http:urlencoded
- http:urlreplacement

Delete Extensibility

Deletes the selected extensibility item.

Service**Insert Service**

Adds a new service in the Services window.

Delete Service

Deletes the selected service in the Services window.

Insert Port

Adds a new port to the selected service in the Services window.

Delete Port

Deletes the selected port from the currently selected service.

21.13.2 WSDL 2.0 Components

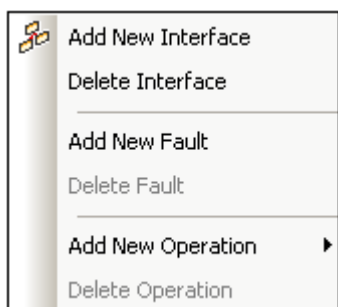
Mousing over the **WSDL 2.0 Components** menu item scrolls out a submenu (*screenshot below*) from which various commands for editing WSDL 2.0 components can be selected.



Each item of the WSDL 2.0 Components menu (*screenshot above*) rolls out its own submenu, from which commands relating to that component can be selected. The commands in each of these submenus are described in the subsections of this section.

Interface

The following commands are available in the **Interface** menu (*screenshot below*).

**Add new interface**

Adds a new interface box to the Interfaces column of the Main Window. The default name of the interface is highlighted in the interface box, enabling you to enter a new name directly.

Delete interface

Deletes the selected interface.

Add new fault

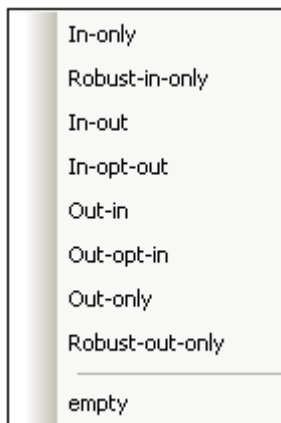
Adds a new `fault` element to the selected interface. The default name of the `fault` is highlighted in the interface box, enabling you to enter a new name directly.

Delete fault

Deletes the selected fault.

Add new operation

Adds a new `operation` element to the selected interface. The type of operation to be added is selected from the pop-out menu (*screenshot below*), and may be one of the operation types shown in the screenshot.



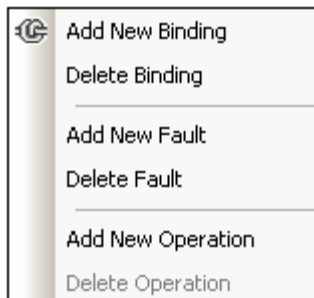
The default name of the `operation` is highlighted in the interface box, enabling you to enter a new name directly.

Delete operation

Deletes the selected operation.

Binding

The following commands are available in the **Binding** menu (*screenshot below*).



Add new binding

Adds a new binding box to the Bindings column of the Main Window. The default name of the binding is highlighted in the binding box, enabling you to enter a new name directly.

Delete binding

Deletes the selected binding.

Add new fault

Adds a new `fault` element to the selected binding. A `fault` element in a binding contains a `ref` attribute that references a fault declared in an interface. In the newly created fault in the binding, the interface fault that is to be referenced can be selected from the combo box of the newly created fault.

Delete fault

Deletes the selected fault.

Add new operation

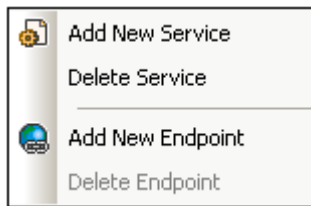
Adds a new `operation` element to the selected binding. An `operation` element in a binding contains a `ref` attribute that references an operation declared in an interface. In the newly created operation in the binding, the interface operation that is to be referenced can be selected from the combo box of the newly created binding operation.

Delete operation

Deletes the selected operation.

Service

The following commands are available in the **Service** menu (*screenshot below*).

**Add new service**

Adds a new service box to the Services column of the Main Window. The default name of the service is highlighted in the service box, enabling you to enter a new name directly. The interface reference can be selected from the combo box for the *Interface* property.

Delete service

Deletes the selected service.

Add new endpoint

Adds a new `endpoint` element to the selected service. The default name of the endpoint is highlighted in the service box, enabling you to enter a new name directly. The binding reference can be selected from the combo box for the *Binding* property. The address of the endpoint must be entered in the field for the *Address* property.

Delete endpoint

Deletes the selected endpoint.

21.13.3 Types, Save Diagram

New Schema

This option only becomes active if the WSDL file does not contain a schema element. Please note that when using the **File | New** menu option, a schema element is included in the skeleton WSDL file. The menu item cannot be selected in this case (see below).

```
<types>  
  <xs:schema/>  
</types>
```

Edit Schema(s) in Schema View

Opens a skeleton schema file if the WSDL file does not contain a reference to a specific schema. This is the case if you have used the **File | New** menu option. If a reference to a specific schema exists, then the schema opens in the embedded Schema View of the graphical WSDL editor.

The **Save Diagram** command saves the design diagram as a PNG file.

21.13.4 Generate Documentation

The **WSDL | Generate Documentation** command generates detailed documentation of the current WSDL file. You can output the documentation as an HTML, MS Word, RTF, or PDF file. The documentation generated by this command can be freely altered and used; permission from Altova to do so is not required. Documentation is generated for components you select in the WSDL Documentation dialog (which appears when you select the Generate Documentation command). Related components are hyperlinked in the onscreen output, enabling you to

navigate from component to component. Note that WSDL documentation is also generated for **imported WSDL files**. The various documentation-generation options are described in the section, [Documentation Options](#).

Note: In order to generate documentation in MS Word format, you must have MS Word (version 2000 or later) installed.

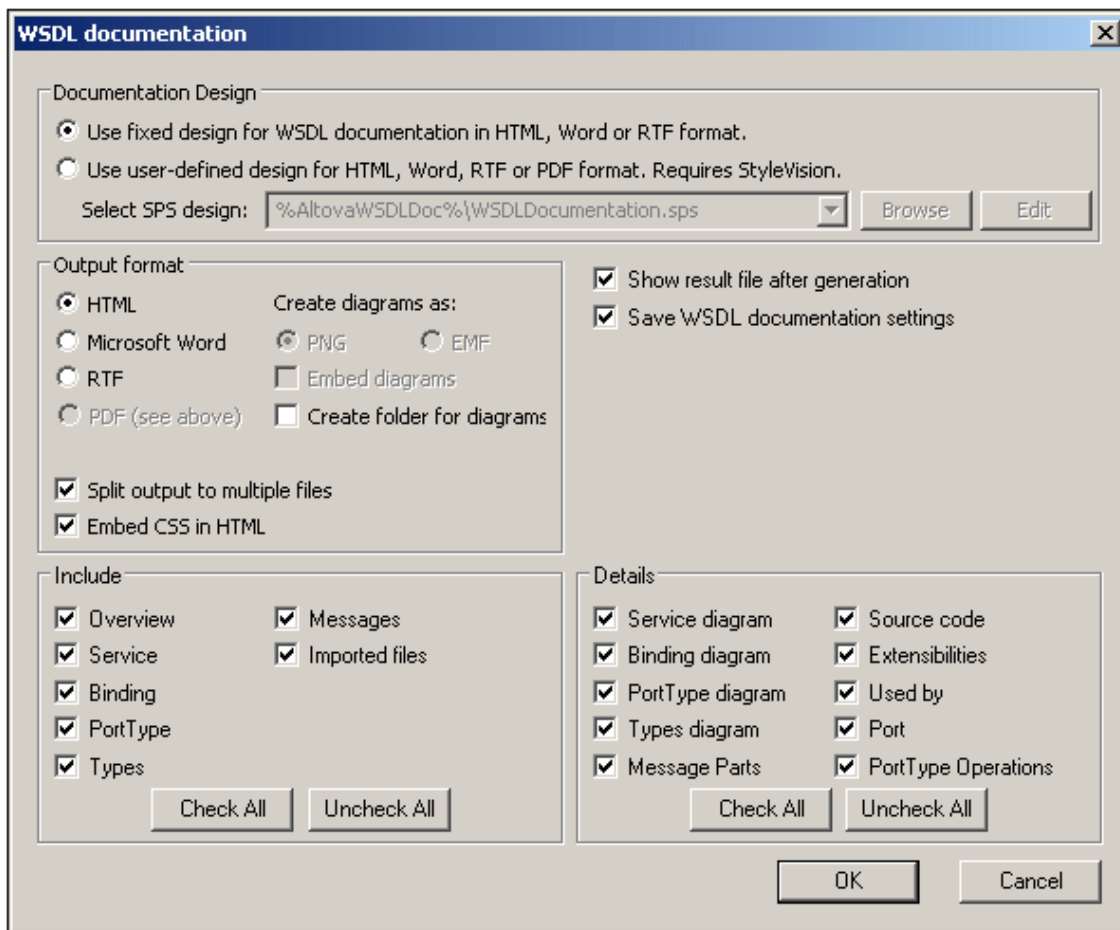
You can either use XMLSpy's fixed design for the generated document, or you can use a StyleVision SPS for the design. Using a StyleVision SPS enables you to customize the design of the generated documentation. How to do this is explained in the section, [User-Defined Design](#).

Note: In order to use an SPS to generate WSDL documentation, you must have StyleVision installed on your machine.

Documentation Options

The **WSDL | Generate Documentation** command pops up the WSDL Documentation dialog (*screenshot below*), in which you can select options for the documentation.

In the Documentation Design pane of the dialog you can select whether to use the fixed XMLSpy design for the generated documentation or whether to use a customized design created in a StyleVision SPS. Select the option you want. Note that PDF output is available only for documentation generated with a StyleVision SPS, not for documentation generated using a fixed design. How to work with a user-defined design is described in the section, [User-Defined Design](#).



The other options in the WSDL Documentation dialog are explained below. Depending on whether a WSDL 1.1 document or a WSDL 2.0 document is active, the items in the Include and Details pane of the dialog will be different. The screenshot above shows the WSDL Documentation dialog for a WSDL 1.1 document.

- The required format is specified in the Output Format pane: either HTML, Microsoft Word, RTF, or PDF. (The PDF output format is only available if you use a StyleVision SPS to generate the documentation.) On clicking **OK**, you will be prompted for the name of the output file and the location to which it should be saved.
- Microsoft Word documents are created with the .doc file extension when generated using a fixed design, and with a .docx file extension when generated using a StyleVision SPS.
- The documentation can be generated either as a single file or be split into multiple files. When multiple files are generated, each file corresponds to a component. What components are included in the output is specified using the check boxes in the Include pane. In fixed designs, links between multiple documents are created automatically.
- For HTML output, the CSS style definitions can be either saved in a separate CSS file or embedded in the HTML file (in the <head> element). If a separate CSS file is created, it will be given the same name as the HTML file, but will have a .css extension. Check or uncheck the *Embed CSS in HTML* check box to set the required option.
- The *Embed Diagrams* option is enabled for the MS Word, RTF, and PDF output options. When this option is checked, diagrams are embedded in the result file, either in PNG or EMF format. Otherwise diagrams are created as PNG or EMF files, which are displayed in the result file via object links.

- When the output is HTML, all diagrams are created as document-external PNG files. If the *Create folder for diagrams* check box is checked, then a folder will be created in the same folder as the HTML file, and the PNG files will be saved inside it. This folder will have a name of the format *HTMLFilename diagrams*. If the *Create folder for diagrams* check box is unchecked, the PNG files will be saved in the same folder as the HTML file.
- In the Include pane, you select which items you want to include in the documentation. The **Check All** and **Uncheck All** buttons enable you to quickly select or deselect all the options in the pane.
- The Details pane lists the details that may be included for each component. Select the details you wish to include in the documentation. The **Check All** and **Uncheck All** buttons enable you to quickly select or deselect all the options in the pane.
- The *Show Result File* option is enabled for all output options. When this option is checked, the result files are displayed in Browser View (HTML output), MS Word (MS Word output), and the default applications for . rtf files (RTF output) and . pdf files (PDF output).

Parameter values

If the StyleVision SPS contains one or more parameter definitions, then on clicking **OK**, a dialog pops up listing all the parameters defined in the SPS. You can enter parameter values in this dialog to override the default parameter values that were assigned in the SPS.

User-Defined Design

Instead of the fixed standard XMLSpy design, you can create a customized design for the WSDL documentation. The customized design is created in a StyleVision SPS, which is a design template for the output document.

Creating the SPS

A StyleVision Power Stylesheet (or SPS) is created using [Altova's StyleVision](#) product. An SPS for generating WSDL documentation must be based on an XML Schema that specifies the structure of the WSDL documentation. Two schemas, one for WSDL 1.1 and the second for WSDL 2.0, are delivered with your XMLSpy package. They are, respectively, *WSDLDocumentation.xsd* and *WSDL20Documentation.xsd*, located respectively in the folders of the [\(My Documents\) folder](#):

- `C:\Documents and Settings\\My Documents\Altova\XMLSpy2012\Documentation\WSDL.`
- `C:\Documents and Settings\\My Documents\Altova\XMLSpy2012\Documentation\WSDL20.`

When creating the SPS design in StyleVision, nodes from the schema are placed in the design template and assigned styles and properties. Additional components, like links, tables and images, can also be added to the SPS design. In this way, the entire output document can be designed in the SPS. How to create an SPS design in StyleVision is described in detail in the StyleVision user manual.

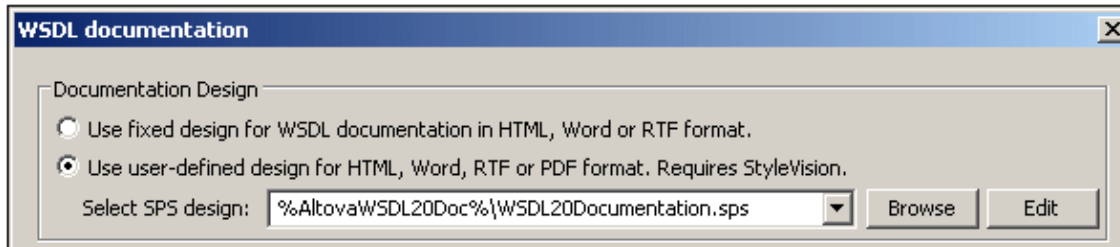
The advantage of using an SPS for generating WSDL documentation is that you have complete control over the SPS design. Note also that PDF output of the WSDL documentation is available only if a user-defined SPS is used; PDF output is not available if the fixed XMLSpy design is used.

Specifying the SPS to use for WSDL documentation

After an SPS has been created, it can be used to generate WSDL documentation. The SPS you

wish to use for generating the WSDL documentation is selected in the WSDL Documentation dialog (accessed via the **WSDL | Generate Documentation** command). In the Documentation Design pane of this dialog (*screenshot below*), select the *Use User-Defined Design* radio button. You can then click the **Browse** button and browse for the SPS you want. Click the dialog's **OK** button, and, in the Save dialog that pops up, select the folder for, and enter the name of, the output file.

Note: The SPS file must correctly locate the schema on which it is based:
WSDLDocumentation.xsd **or** WSDL20Documentation.xsd (*see above*).



Two editable SPS designs, one each for WSDL 1.1 and WSDL 2.0, are delivered with XMLSpy. They are, respectively, in the `WSDL` and `WSDL20` sub-folders of the [\(My Documents\) folder](#): `C:\Documents and Settings\<username>\My Documents\Altova\XMLSpy2011\Documentation\`. They are named:

- `WSDL\WSDLDocumentation.sps`
- `WSDL20\WSDL20Documentation.sps`

These files, together with other SPS files you have recently browsed for, will be available in the combo box of the *Use User-Defined* option (*see screenshot above*).

Clicking the **Edit** button in the Documentation Design pane launches StyleVision and opens the selected SPS in a StyleVision window. In order to preview the result document in StyleVision, you will need a Working XML file. Sample XML files for this purpose, called `TimeService.xml` and `TimeService20.xml`, are supplied with your application and are located in the [\(My Documents\) folder](#):

`C:\Documents and Settings\<username>\My Documents\Altova\XMLSpy2012\Documentation\WSDL(20)\SampleData`

Note: In order to use an SPS to generate WSDL documentation, you must have StyleVision installed on your machine.

21.13.5 Reparse WSDL Document

Reparses the document. This is required in some situations, for example, when a schema associated with the WSDL document has been modified. Reparsing the WSDL document will update it with information from the modified schema document.

21.13.6 Convert to WSDL 2.0

The **Convert to WSDL 2.0** command is enabled only when a WSDL 1.1 is active in WSDL View. It generates a WSDL 2.0 document from the active WSDL 1.1 document. Clicking this command pops up a File Save dialog, in which you can specify the location and name of the WSDL 2.0 file that will be generated by XMLSpy.

On clicking **OK** in the File Save dialog, a WSDL 2.0 document is generated and saved to the specified location, and opened in WSDL View in a new tab. The file can then be edited as

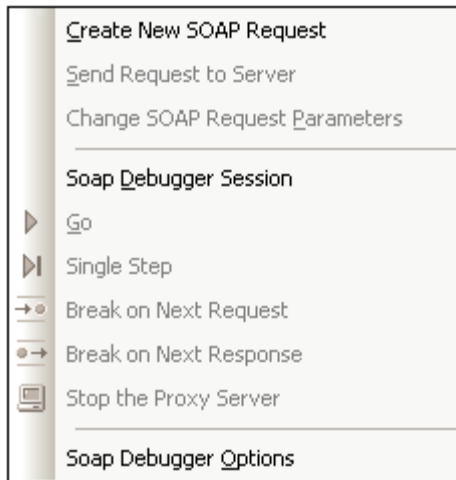
required, just like any other WSDL 2.0 document.

21.13.7 Generate WSDL Program Code with MapForce

The **Generate WSDL Program Code with MapForce** command launches Altova's MapForce if the application is installed. MapForce enables you to map a schema to another DTD, XML Schema, or database, to generate XML, and to generate program code from the WSDL file.

21.14 SOAP Menu

XMLSpy supports SOAP versions 1.1 and 1.2, and WSDL versions 1.1 and 2.0.



The [SOAP: How To](#) section that follows the menu descriptions, shows you how to use the SOAP debugger using the **nanonull.com timeservice** server supplied by Altova. Please use this service to test the SOAP debugger. The AirportWeather web service, described on the following pages, might not always be available to you.

Use the SOAP functionality:

- To test your web services without having to implement client applications
- For quick testing of third party web services

For more info on these specifications please see:

SOAP <http://www.w3.org/TR/SOAP/>

WSDL <http://www.w3.org/TR/wsdl>

21.14.1 Create New SOAP Request

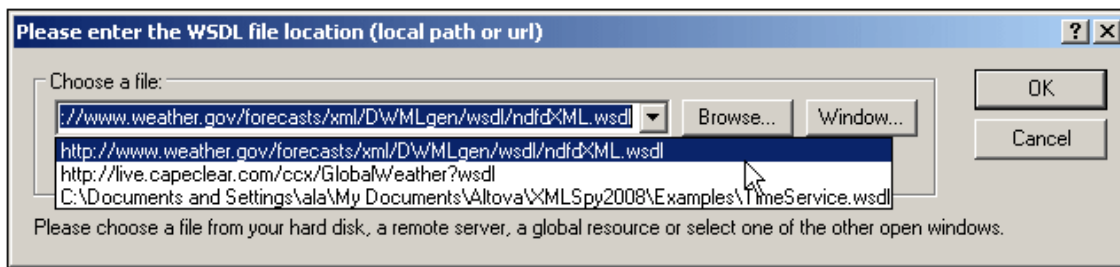
This command creates a new SOAP request document. It involves the following steps:

1. [Enter the WSDL file location and connect to the server.](#)
2. The server responds with a list of operations. [Select the SOAP operation you want.](#)
3. The server responds with a SOAP Request form in XML format. [Define the SOAP Request form.](#)

We demonstrate the process below by creating a SOAP request for the US National Digital Forecast Database (NDFD) SOAP Service (<http://www.nws.noaa.gov/xml/>).

Connecting to the SOAP server

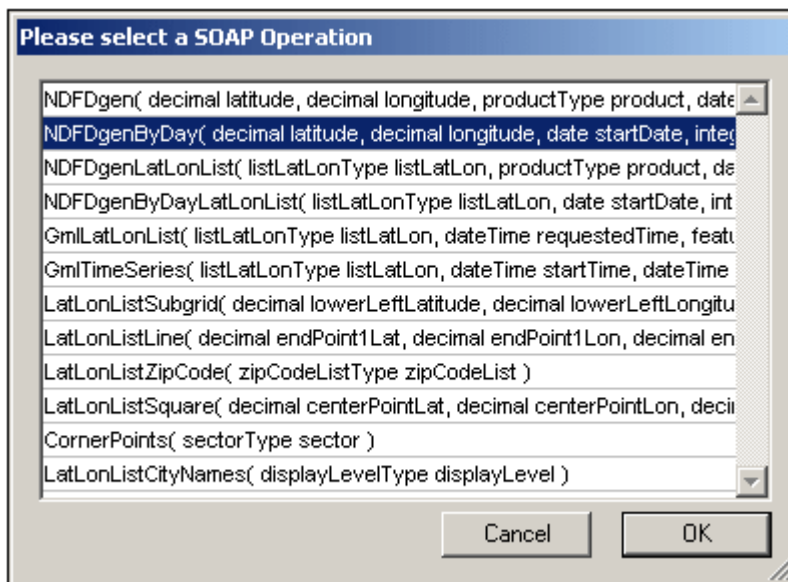
The connection is made via a WSDL file. In our example, the URI of the WSDL file is: <http://www.weather.gov/forecasts/xml/DWMLgen/wsdl/ndfdXML.wsdl>. To make the connection, click the Create New SOAP Request command, and enter the file URI in the dialog that appears (*screenshot below*).



Click **OK** to confirm the selection.

Select the required SOAP operation

The server responds with a list of operation which are displayed in a dialog (*screenshot below*).



Select an operation and click **OK**. We selected the `NDFDgenByDay` operation.

Define the SOAP Request

The server responds by sending an XML file, which is displayed in the Text View of XMLSpy. For the operation we selected, we received the following XML file.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="
http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:m0="http://www.weather.gov/forecasts/xml/DWMLgen/schema/DWML.xsd">
  <SOAP-ENV:Body>
    <m:NDFDgenByDay xmlns:m="
http://www.weather.gov/forecasts/xml/DWMLgen/wsd/ndfdXML.wsd"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <latitude xsi:type="xsd:decimal">0.0</latitude>
      <longitude xsi:type="xsd:decimal">0.0</longitude>
      <startDate xsi:type="xsd:date">1967-08-13</startDate>
      <numDays xsi:type="xsd:integer">0</numDays>
      <format xsi:type="m0:formatType">String</format>
    </m:NDFDgenByDay>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

We filled in the parameters as required by the XML (*in bold below*):

```
<SOAP-ENV:Body>
  <m:NDFDgenByDay xmlns:m="
http://www.weather.gov/forecasts/xml/DWMLgen/wsd1/ndfdXML.wsd1 "
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <latitude xsi:type="xsd:decimal">45</latitude>
    <longitude xsi:type="xsd:decimal">-90</longitude>
    <startDate xsi:type="xsd:date">2009-10-13</startDate>
    <numDays xsi:type="xsd:integer">1</numDays>
    <format xsi:type="m0:formatType">24 hourly</format>
  </m:NDFDgenByDay>
</SOAP-ENV:Body>
```

This completes the **definition** of this SOAP request. In the next step, we shall [send the request](#).

21.14.2 Send Request to Server

This command sends a SOAP request to the SOAP server. The SOAP request is an XML document. It must be the active document in XMLSpy when the command is selected. How to define a SOAP request is described in the description of the command [Create New SOAP Request](#).

After the SOAP request is sent, a response is received from the SOAP server. This response is an XML document, which is displayed in the Text View of XMLSpy. For example, show below is a screenshot of the XML document that was returned in response to the SOAP request we defined in the section [Create New SOAP Request](#).



```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SO
3 http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:
4 http://www.w3.org/2001/XMLSchema-instance" xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/enc
5 <SOAP-ENV:Body>
6 <ns1:NDFDgenByDayResponse xmlns:ns1="http://www.weather.gov/forecasts/xml/DWMLgen/wsd1/n
7 <dwmlByDayOut xsi:type="xsd:string"><?xml version="1.0"?>
8 <dwml version="1.0" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/
9 xsi:noNamespaceSchemaLocation="http://www.nws.noaa.gov/forecasts/xml/DWMLgen/schema/DWML.xsd">
10 <head>
11 <data>
12 <location>
13 <location-key>point1</location-key>
14 <point latitude="45.00" longitude="-90.00"/>
15 </location>
16 <moreweatherInformation applicable-location="point1">
17 http://forecast.weather.gov/MapClick.php?textField1=45.00&textField2=-90.00
18 </moreweatherInformation>
19 <time-layout time-coordinate="local" summarization="24hourly">
20 <time-layout time-coordinate="local" summarization="12hourly">
21 <parameters applicable-location="point1">
22 <temperature type="maximum" units="Fahrenheit" time-layout="k-p24h-n1-1">
23 <name>Daily Maximum Temperature</name>
24 <value>45</value>
25 </temperature>
26 <probability-of-precipitation type="12 hour" units="percent" time-layout="k-p12h-n2-2">
27 <name>12 Hourly Probability of Precipitation</name>
28 <value>14</value>
29 <value xsi:nil="true"/>
30 </probability-of-precipitation>
31 </parameters>
32 </data>
33 </dwml>
34 </dwmlByDayOut>
35 </ns1:NDFDgenByDayResponse>
36 </SOAP-ENV:Body>
37 </SOAP-ENV:Envelope>
```

Saving and reusing a SOAP request

XMLSpy allows you to save a SOAP request and resend it at a later time. Do this as follows:

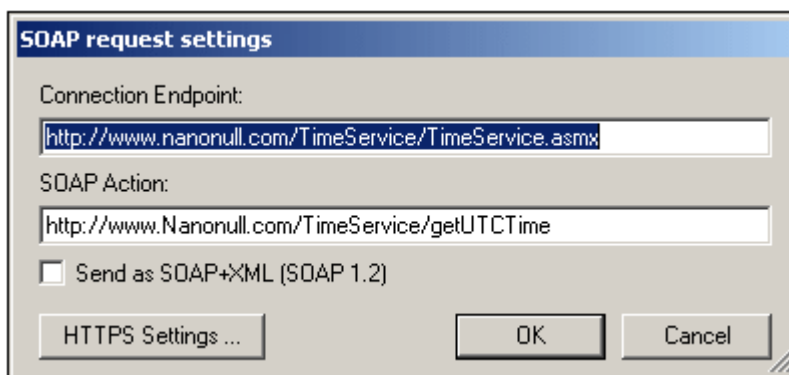
1. Save the SOAP request XML document (**File | Save as**).

2. Close the SOAP request file.
3. Reopen the SOAP request XML document, and select the menu option **SOAP | Send Request to Server**. (Any XML file can be used as a SOAP request document.)

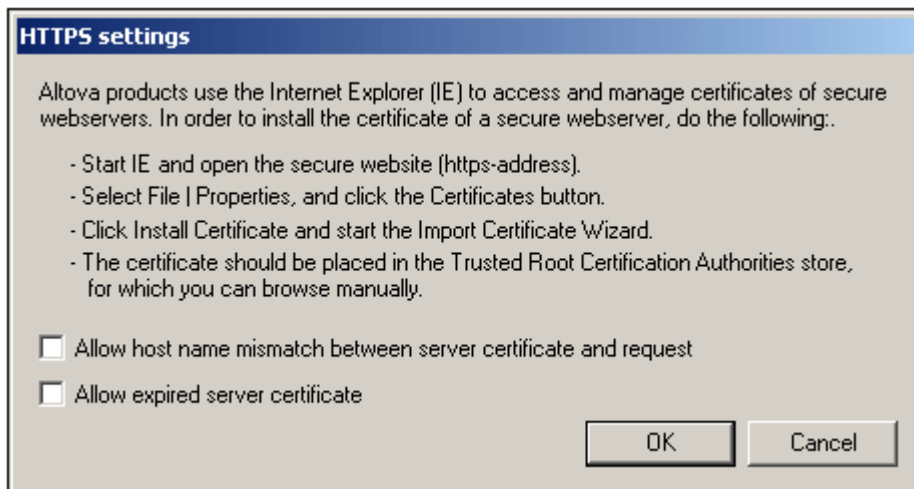
21.14.3 Change SOAP Request Parameters

This command allows you to change the currently defined SOAP parameters of the [SOAP request](#) document. To change a SOAP request, do the following:

1. Make the SOAP request document active.
2. Select the menu option **Soap | Change SOAP request parameters**. This opens the SOAP Request Settings dialog box (*screenshot below*).



3. You can now manually enter the desired Connection Endpoint or SOAP Action settings. If you wish to send the request as SOAP 1.2, check the check box at the bottom of the dialog. If you are using the `https` protocol, you might wish to allow a host name mismatch or an expired server certificate. To make these settings, click the **HTTPS Settings** button. The HTTPS Settings dialog (*screenshot below*) appears.



4. In the HTTPS Settings dialog, you can allow that the host name given in the SOAP request need not match the host name in the server certificate; do this by checking the *Allow host name mismatch* check box. If you wish to allow the connection despite an expired server certificate, then check the *Allow expired server certificate* check box. Confirm the changes with **OK**.

About trusted certificates

Altova products use Internet Explorer (IE) to access and manage trusted certificates of secure webserver. Installing the certificate of a webserver in IE allows IE to access the webserver without issuing a warning or aborting the process. In order to install the certificate of a secure webserver, do the following:

- In Internet Explorer 8, open the secure website.
- Select **File | Properties**, and click the Certificates button.
- Click **Install Certificate** and start the Import Certificate Wizard. (This Wizard can also be accessed via **Tools | Internet Options | Content | Certificates | Import.**)
- The certificate should be placed in the Trusted Root Certification Authorities store, for which you can browse manually.
- Finish the Wizard steps, close the Certificates and Properties dialogs respectively by clicking **OK**. You might need to restart Internet Explorer.

Note: Only change the SOAP action settings if a full list of the SOAP methods and their corresponding SOAP actions are available to, and accessible by, you.

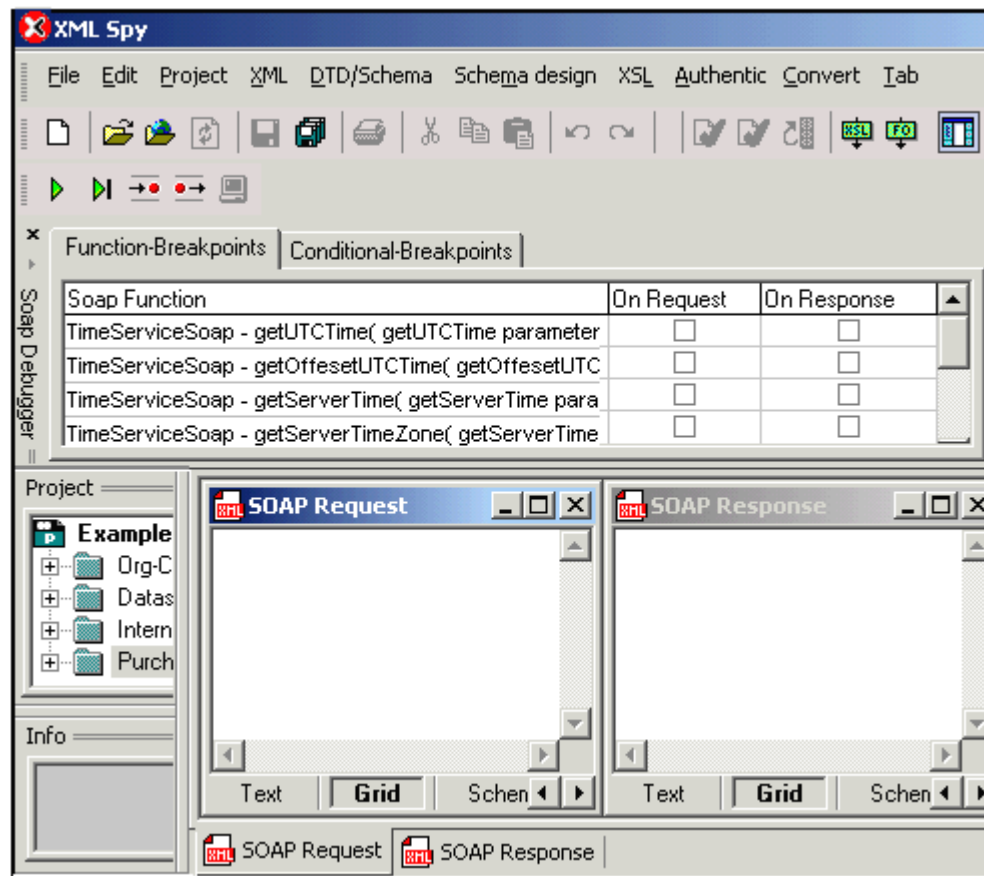
21.14.4 Soap Debugger Session

This command starts a SOAP debugger session.

- A dialog box is immediately opened after you select this command. You then have to select a WSDL file location, generally a URL. You can also select a file via a global resource (click the  icon and select a global resource in the dialog that pops up) or a file in one of the open windows in XMLSpy.
- Select the source and target ports needed for the debugger proxy server and the web service, in the following dialog box.

This opens the SOAP debuggers proxy server in its **inactive** state. Clicking one of the SOAP toolbar icons, starts the SOAP debugger and waits for the client requests.

Please see the [Soap - How to](#) section for a more detailed description.



21.14.5 Go



This command activates the SOAP proxy server and processes the WSDL file until a breakpoint is encountered. The respective SOAP document then appears in one of the SOAP document windows.

21.14.6 Single Step



This command allows you to single-step through the incoming and outgoing SOAP requests and responses. The SOAP debugger stops for each request and response. The proxy server is also started if it was inactive.

21.14.7 Break on Next Request



This command causes the debugger to stop on the next SOAP request, and display the data in the SOAP Request document window. You can directly edit the data in this window before

sending it on to the web service.

21.14.8 Break on Next Response



This command causes the debugger to stop on the next SOAP Response, and display the data in the SOAP Response document window. You can directly edit the data in this window before sending it on to the client.

21.14.9 Stop the Proxy Server



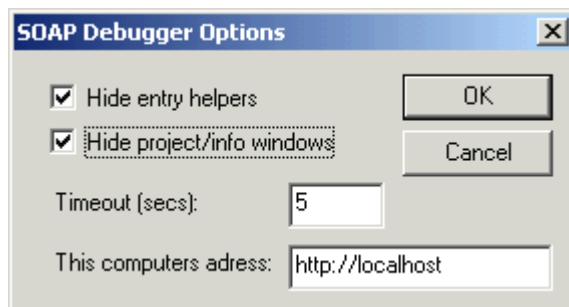
This command stops the debugger proxy server.

21.14.10 Soap Debugger Options

This command allows you to define the SOAP debugger options. Please define these settings **before** you start the SOAP debugger.

Hide

You can hide the entry helpers as well as the project or info windows, enabling you to see more of the data in the SOAP documents.



Timeout (secs)

Most clients have a timeout setting in the data receiving function. The time the debugger remains in a breakpoint is per default 5 seconds. A data packet (http1.1 100 Continue") message is sent to the client once every timeout period you define.

Setting the timeout period to zero suppresses the data packet; you remain in the breakpoint for as long as you want.

This computer's address

Address of the computer on which the SOAP debugger is running. This computer must be accessible to the Client computer.

Enter the address in the form **http://computer name**, where "computer name" is the web address or IP address of the computer on which the SOAP debugger is running. Do not enter a port number in the address. The "localhost" entry always points to your local computer.

21.15 XBRL Menu

The XBRL menu contains commands that are enabled when a taxonomy is being edited in XBRL View. These commands are:

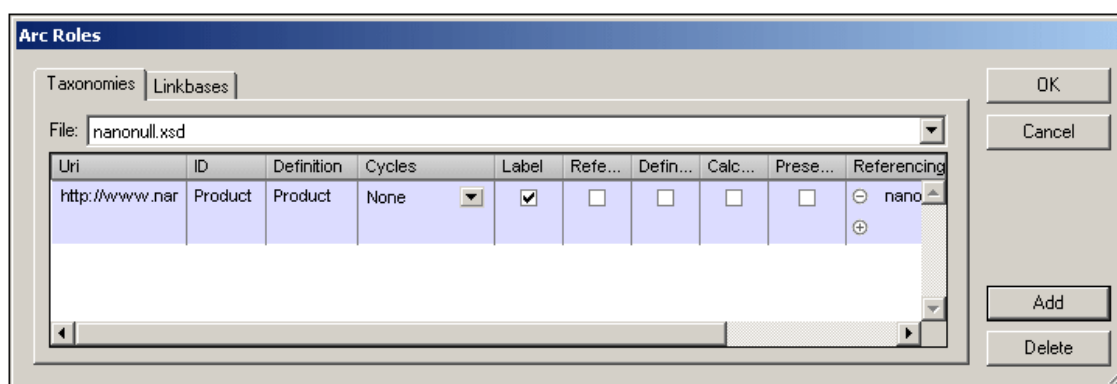
- [Arcroles](#): defines arcroles
- [Linkroles](#): defines linkroles
- [Namespace Prefixes](#): manages taxonomy namespaces
- [Set Target Namespace](#): defines and declares the target namespace of the taxonomy
- [Import/Reference](#): imports an XBRL taxonomy or references a linkbase
- [Generate Documentation](#): generates documentation of the current XBRL taxonomy
- [View Settings](#): sets default for XBRL View
- [Generate XBRL from DB, Excel, CSV with MapForce](#): launches Altova MapForce for generating an XBRL instance file
- [Present XBRL as HTML/PDF/Word with StyleVision](#): launches Altova StyleVision for designing an XBRL report

For more information about XBRL, see the sections [XBRL](#) and [Editing Views | XBRL View](#).

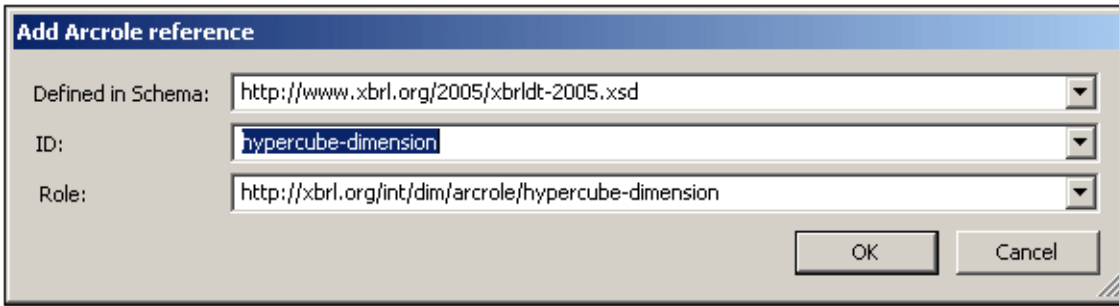
21.15.1 Arcroles

The **Arcroles** command pops up the Arc Roles dialog (*screenshot below*) in which arcroles can be created for a taxonomy. Arcroles are stored in the concept definitions file, within the `appinfo` element. They specify the role of an arc.

In the **Taxonomies** tab of the Arc Roles dialog (*screenshot below*), only taxonomies that are editable or that contain an arcrole or linkrole are listed in the combo box. You can add an arcrole to a taxonomy by clicking the **Add** button. Then define the arcrole's URI, ID, definition, and cycles. To specify in which kinds of relationships the arcrole should be available, check the boxes of the required relationship kinds. Linkbases that reference an arcrole can be added to the Referencing Linkbase Files column.



The **Linkbases** tab provides another view of the taxonomy's arcroles. In this view, you add and view arcroles according to individual linkbases (for example calculation or presentation linkbases). Select a linkbase in the combo box and then add or delete an arcrole as required. When you click the **Add** button, the Add Arcrole Reference dialog (*screenshot below*) pops up.



Add Arcrole reference

Defined in Schema:

ID:

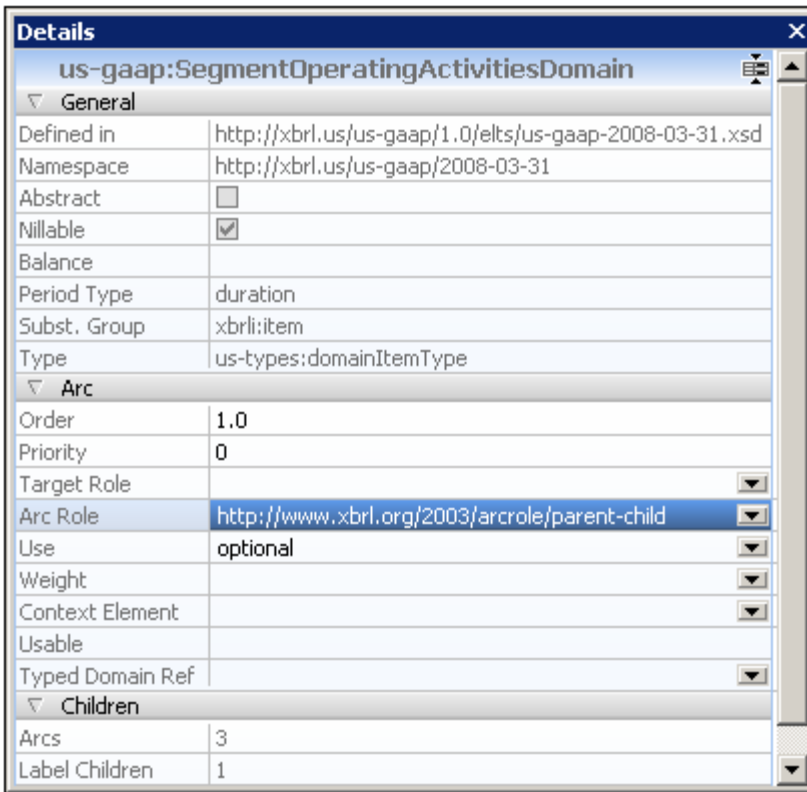
Role:

OK Cancel

Each of the entries in this dialog is a combo box that enables you to select from available options. The *Defined in Schema* field enables you to select the taxonomy in which the arcrole is defined. The *ID* and *Role* combo boxes provide the available arcroles. After you have selected an arcrole and clicked **OK**, the reference is added to the linkbase. In the Taxonomies tab, the arcrole you referenced will show the referencing linkbase in the Referencing Linkbase Files column. If you wish to make this linkrole available for a particular kind of relationship, you will still, however, have to check the appropriate relationship kind check box.

After an arcrole has been created in the taxonomy it can be used to specify the role of an arc in a relationship kind for which the arc is available according to its definition. In the screenshot above, for example, the arcrole has been made available for arcs in label relationships.

The arcrole of an arc is selected in the Details entry helper (*screenshot below; arcrole highlighted*).



Details

us-gaap:SegmentOperatingActivitiesDomain

General

Defined in:

Namespace:

Abstract: ☐

Nilable: ☒

Balance:

Period Type:

Subst. Group:

Type:

Arc

Order:

Priority:

Target Role:

Arc Role:

Use:

Weight:

Context Element:

Usable:

Typed Domain Ref:

Children

Arcs:

Label Children:

With the element at the end of an arc selected, in the Details Entry Helper, select the required item from the dropdown list of the arcrole entry.

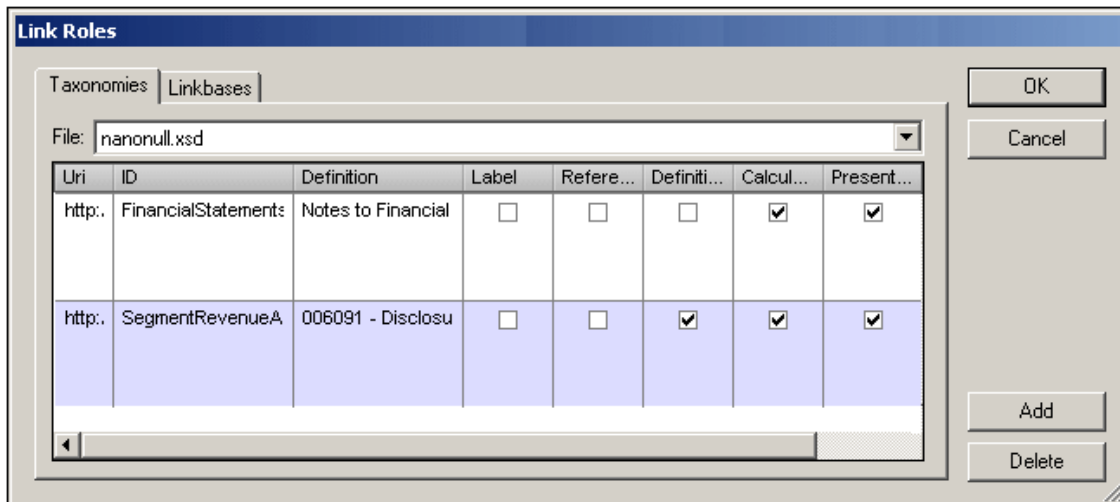
21.15.2 Linkroles

The **Linkroles** command pops up the Link Roles dialog (*screenshot below*) in which linkroles can be created for a taxonomy. Linkroles are stored in the concept definitions file, within the `appinfo` element (*see listing below*). Linkroles are used not only in `definitionLink` elements but also in the containing elements of other relationship kinds (for example, in `calculationLink` and `presentationLink` elements).

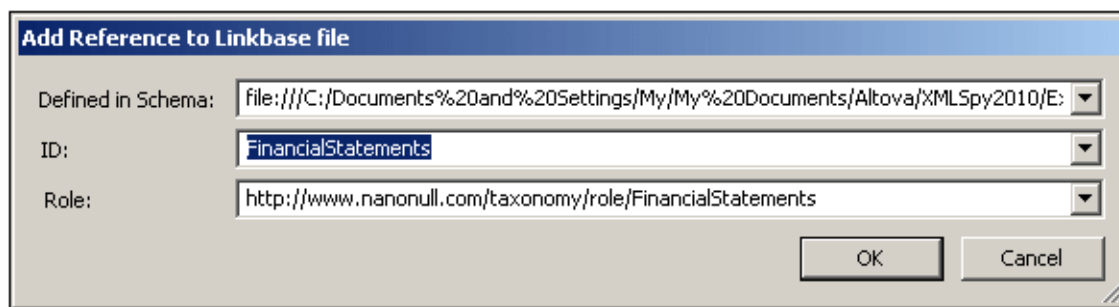
```
<xs:appinfo>
  <link:roleType id="SegmentRevenueAndOperatingIncome"
    roleURI="
http://www.nanonull.com/taxonomy/role/SegmentRevenueAndOperatingIncome">
    <link:definition>006091 - Disclosure - Segment Revenue and Operating
Income</link:definition>
    <link:usedOn>link:calculationLink</link:usedOn>
    <link:usedOn>link:definitionLink</link:usedOn>
    <link:usedOn>link:presentationLink</link:usedOn>
  </link:roleType>
</xs:appinfo>
```

In the listing above, notice that there are `usedOn` elements that specify in which kind of relationships this linkrole may be used.

In the **Taxonomies** tab of the Link Roles dialog (*screenshot below*), you can add a linkrole to a taxonomy file by clicking the **Add** button. Then define the linkrole's URI and ID (*refer to listing above*). To specify for which kinds of relationships a linkrole should be available, check the boxes of the required relationship kinds. In the Referencing Linkbase Files column, for each linkrole, you can add or delete the linkbase files that reference the linkrole.



The **Linkbases** tab provides another view of the taxonomy's linkroles. In this view, you add and view linkroles according to individual linkbases (for example calculation or presentation linkbases). Select a linkbase in the combo box and then add or delete a linkrole as required. For example, you could add a linkrole to the calculation linkbase. When you click the **Add** button, the Add Reference to Linkbase File dialog (*screenshot below*) pops up.

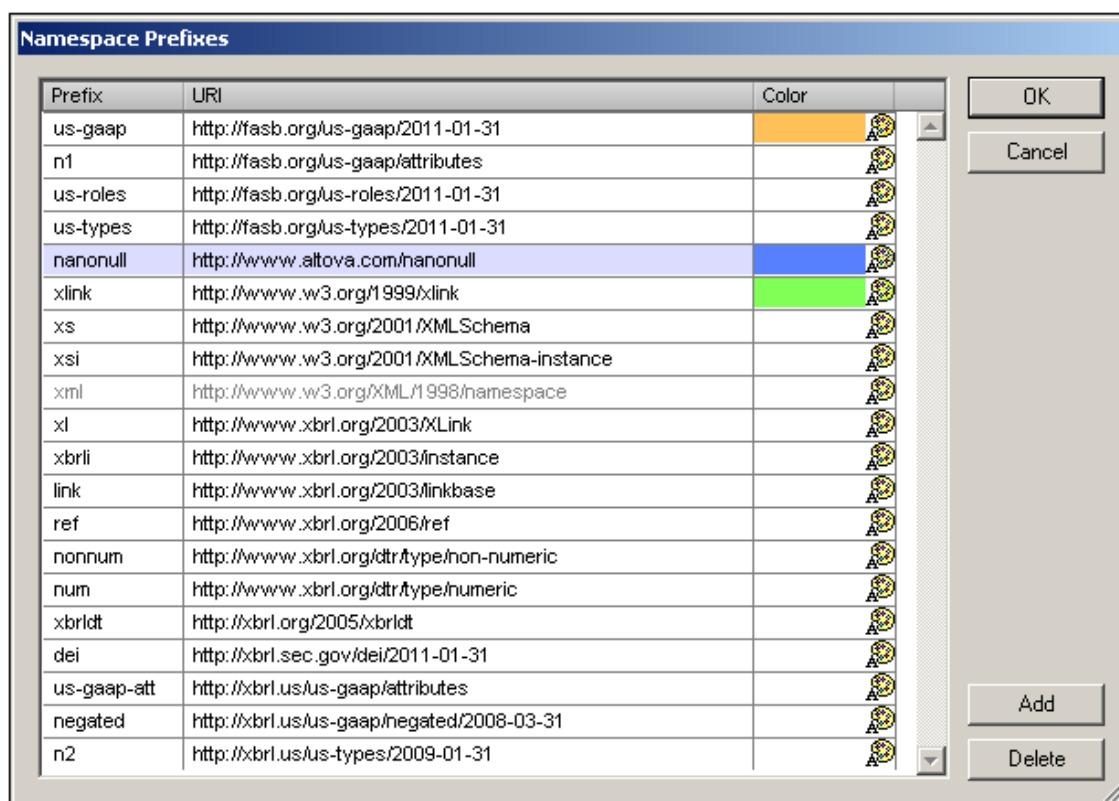


Each of the entries in this dialog is a combo box that enables you to select from available options. The *Defined in Schema* field enables you to select the taxonomy in which the linkrole is defined. The *ID* and *Role* combo boxes provide the available linkroles. After you have selected a linkrole and clicked **OK**, the reference is added to the linkbase. In the Taxonomies tab, the linkrole you referenced will show the referencing linkbase in the Referencing Linkbase Files column. If you wish to make this linkrole available for a particular kind of relationship, you will still, however, have to check the appropriate relationship kind check box.

After a linkrole has been created in the taxonomy it is used when [creating relationships](#).

21.15.3 Namespace Prefixes

The **Namespace Prefixes** command pops up the Namespace Prefixes dialog (*screenshot below*), which displays all the namespaces in the taxonomy, including those of imported taxonomies. In the Namespace Prefixes dialog you can edit namespaces and prefixes, and set background colors for individual namespaces. When a background color is set for a namespace, elements in that namespaces appear in the Main Window and entry helpers with that background color. Note that a color setting for a given namespace applies for that namespace across all taxonomy documents opened in XBRL View.

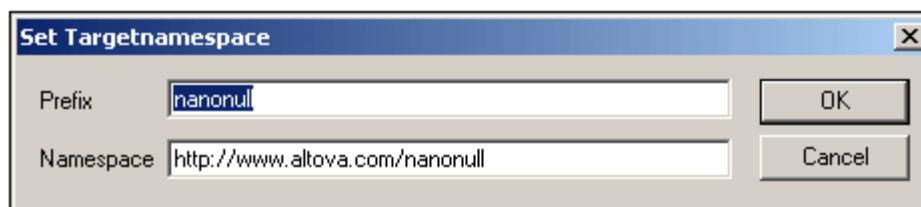


To add or delete a namespace, use the **Add** or **Delete** buttons, respectively. A color is assigned to a namespace via the color palette for that namespace. When you are done with editing in the Namespaces dialog, click **OK** to finish.

The **target namespace** of the taxonomy is also listed in the Namespaces dialog. The target namespace, however, should not be modified in this dialog, but via the [Set Target Namespace](#) command. For more information on target namespaces, see the [XBRL section of the documentation](#).

21.15.4 Set Target Namespace

The **Set Target Namespace** command enables a target namespace to be set for a taxonomy document. Clicking the command pops up the Set Target Namespace dialog (*screenshot below*). In it you can enter the desired target namespace and a prefix for it. Click **OK** to finish.



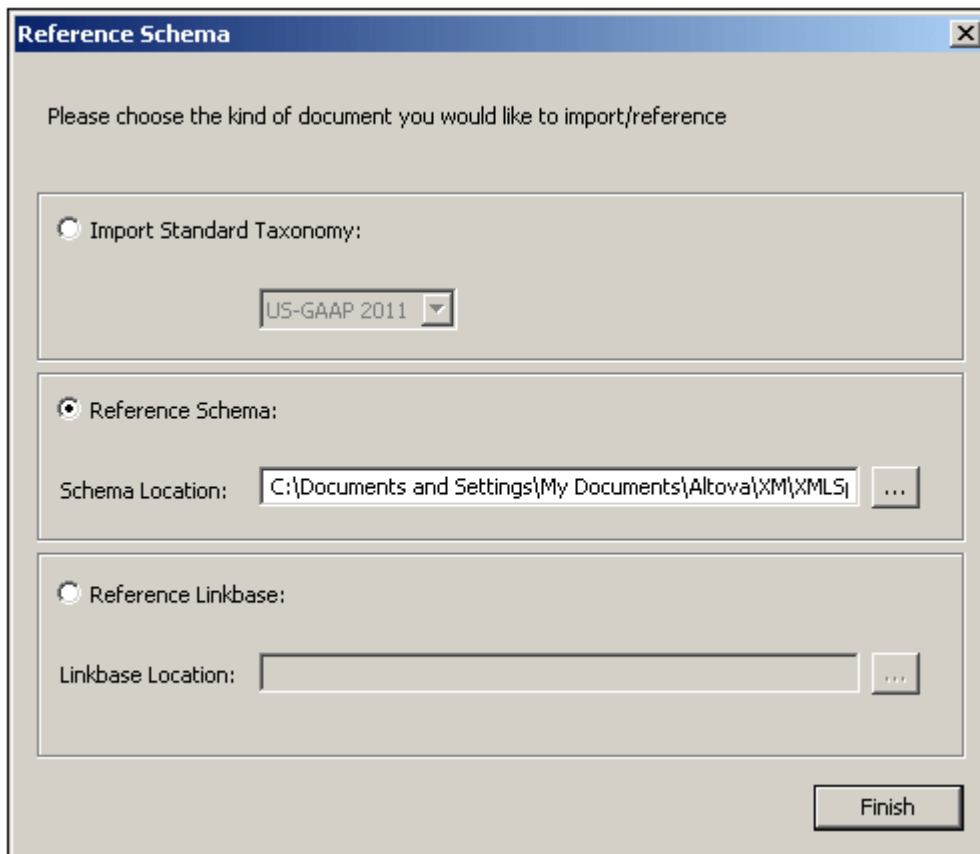
The target namespace will be defined and it will also be declared:

```
<xs:schema targetNamespace="http://www.altova.com/XBRL/Taxonomies"
  xmlns:ns1="http://www.altova.com/XBRL/Taxonomies" >
  ...
</xs:schema>
```

In the listing above, the target namespace is defined with the `targetNamespace` attribute and it is then declared with a prefix of `ns1`.

21.15.5 Import/Reference

The **Import/Reference** command pops up a dialog (*screenshot below*) in which you can specify the schema to import or the linkbase to reference.



The dialog provides the following radio button options:

- *Importing a standard taxonomy:* This option enables you to quickly and correctly import a US-GAAP taxonomy or an IFRS taxonomy. Select the required standard taxonomy from the dropdown menu of the combo box and click Next. This takes you to the respective next dialog, for completing your specification of the required taxonomy. The process is described in the section, [Creating a New Taxonomy](#).
- *Importing any taxonomy (Reference Schema):* This option enables you to import any taxonomy by specifying the location of the taxonomy file (.xsd file).

```
<xs:import namespace="http://www.altova.com/nanonu11 "
schemaLocation="XBRL%20Examples/Nanonu11/nanonu11.xsd"/>
```
- *Referencing a linkbase:* A linkbase can be specified for inclusion in the taxonomy. Do this by specifying the location of the linkbase file and clicking **Finish**. A reference to the linkbase file is created in the taxonomy. The relationship type of the newly referenced linkbase can then be specified right-clicking the filename and selecting the [Set Linkbase Kind](#) command.

21.15.6 Generate Documentation

The **XBRL | Generate Documentation** command generates detailed documentation of the current XBRL taxonomy. You can output the documentation as an HTML, MS Word, RTF, or PDF file. The documentation generated by this command can be freely altered and used; permission from Altova to do so is not required. Documentation is generated for components you select in the XBRL Taxonomy Documentation dialog (which appears when you select the Generate Documentation command). Related components are hyperlinked in the onscreen output, enabling you to navigate from component to component. The various documentation-generation options are described in the section, [Documentation Options](#).

Note: In order to generate documentation in MS Word format, you must have MS Word (version 2000 or later) installed.

You can either use XMLSpy's fixed standard design for the generated document, or you can use a StyleVision SPS for the design. Using a StyleVision SPS enables you to customize the design of the generated documentation as well as to generate PDF as an additional output format. How to work with an SPS this is explained in the section, [User-Defined Design](#).

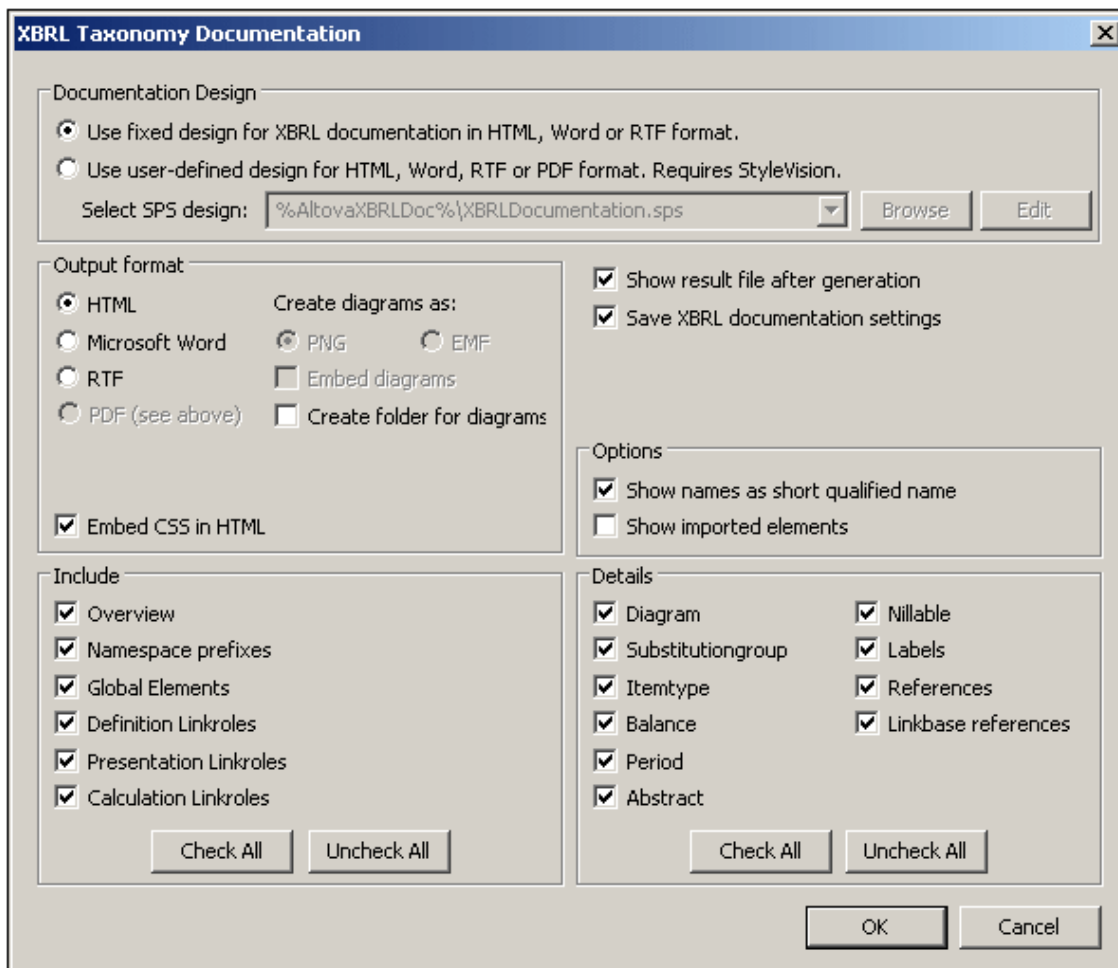
Note: In order to use an SPS to generate schema documentation, you must have StyleVision installed on your machine.

Documentation Options

The **XBRL | Generate Documentation** command pops up the XBRL Taxonomy Documentation dialog (*screenshot below*), in which you can select options for the documentation.

In the Documentation Design pane of the dialog you can select whether to use the fixed standard XMLSpy design for the generated documentation or whether to use a customized design created in a StyleVision SPS. Select the option you want. Note that PDF output is available only for documentation generated with a StyleVision SPS, not for documentation generated using a fixed design. How to work with a user-defined design is described in the section, [User-Defined Design](#).

Note: In order to use an SPS to generate schema documentation, you must have StyleVision installed on your machine.



Clicking the **Generate Documentation** command opens the XBRL Taxonomy Documentation dialog:

- The required format is specified in the Output Format pane: either HTML, Microsoft Word, RTF, or PDF. (The PDF output format is only available if you use a StyleVision SPS to generate the documentation.) On clicking **OK**, you will be prompted for the name of the output file and the location to which it should be saved.
- Microsoft Word documents are created with the .doc file extension when generated using a fixed design, and with a .docx file extension when generated using a StyleVision SPS. In order to generate documentation in MS Word format, you must have MS Word (version 2000 or later) installed.
- For HTML output, the CSS style definitions can be either saved in a separate CSS file or embedded in the HTML file (in the <head> element). If a separate CSS file is created, it will be given the same name as the HTML file, but will have a .css extension. Check or uncheck the *Embed CSS in HTML* check box to set the required option.
- The *Embed Diagrams* option is enabled for the MS Word and RTF output options. When this option is checked, diagrams are embedded in the result file, either in PNG or EMF format. Otherwise diagrams are created as PNG or EMF files, which are displayed in the result file via object links.
- When the output is HTML, all diagrams are created as document-external PNG files. If the *Create folder for diagrams* check box is checked, then a folder will be created in the same folder as the HTML file, and the PNG files will be saved inside it. This folder will

have a name of the format `HTMLFilename_diagrams`. If the *Create folder for diagrams* check box is unchecked, the PNG files will be saved in the same folder as the HTML file.

- In the Include pane, you select which items you want to include in the documentation. The *Overview* option lists all components, organized by component type, at the top of the file. The **Check All** and **Uncheck All** buttons enable you to quickly select or deselect all the options in the pane.
- The Details pane lists the details that may be included for each component. Select the details you wish to include in the documentation. The **Check All** and **Uncheck All** buttons enable you to quickly select or deselect all the options in the pane.
- The *Show Result File* option is enabled for all output options. When this option is checked, the result files are displayed in Browser View (HTML output), MS Word (MS Word output), and the default applications for .rtf files (RTF output) and .pdf files (PDF output).
- In the Options pane, you can select (i) whether element name should be shown with just a prefix (short qualified name) or in its expanded form (with the full namespace); and (ii) whether imported elements should also be displayed.

Parameter values

If the StyleVision SPS contains one or more parameter definitions, then on clicking **OK**, a dialog pops up listing all the parameters defined in the SPS. You can enter parameter values in this dialog to override the default parameter values that were assigned in the SPS.

User-Defined Design

Instead of the fixed standard XMLSpy design, you can create a customized design for XBRL taxonomy documentation. The customized design is created in a StyleVision SPS, which is a design template for the output document.

Creating the SPS

A StyleVision Power Stylesheet (or SPS) is created using [Altova's StyleVision](#) product. An SPS for generating XBRL taxonomy documentation must be based on an XML Schema that specifies the structure of the XBRL taxonomy documentation. This schema is called `XBRLDocumentation.xsd`, and it is delivered with your XMLSpy package. It is stored in the folder: `C:\Documents and Settings\<username>\My Documents\Altova\XMLSpy2012\Documentation\XBRL`.

When creating the SPS design in StyleVision, nodes from the `XBRLDocumentation.xsd` schema are placed in the design template and assigned styles and properties. Additional components, like links, tables and images, can also be added to the SPS design. In this way, the entire output document can be designed in the SPS. How to create an SPS design in StyleVision is described in detail in the StyleVision user manual.

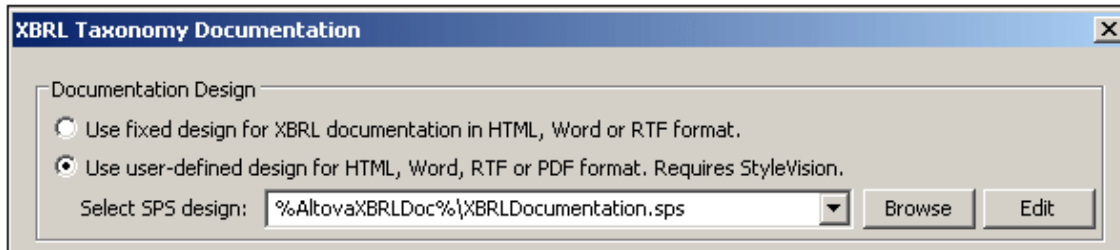
The advantage of using an SPS for generating XBRL taxonomy documentation is that you have complete control over the SPS design. Note also that PDF output of the XBRL taxonomy documentation is available only if a user-defined SPS is used; PDF output is not available if the fixed XMLSpy design is used.

Specifying the SPS to use for XBRL taxonomy documentation

After an SPS has been created, it can be used to generate XBRL taxonomy documentation. The SPS you wish to use for generating the XBRL taxonomy documentation is selected in the XBRL Taxonomy Documentation dialog (accessed via the **XBRL | Generate Documentation** command). In the Documentation Design pane of this dialog (see screenshot below), select the *Use User-Defined Design* radio button. You can then click the **Browse** button and browse for

the SPS you want. Click the dialog's **OK** button, and, in the Save dialog that pops up, select the folder for, and enter the name of, the output file.

Note: The SPS file must correctly locate the schema on which it is based:
XBRLDocumentation.xsd (*see above*).



One editable SPS design for XBRL taxonomy documentation generation is delivered with XMLSpy. It is named `XBRLDocumentation.sps` and is in the folder: `C:\Documents and Settings\<username>\My Documents\Altova\XMLSpy2011\Documentation\XBRL\`. This SPS file, together with other SPS files you have recently browsed for, will be available in the combo box of the *Use User-Defined* option (*see screenshot above*).

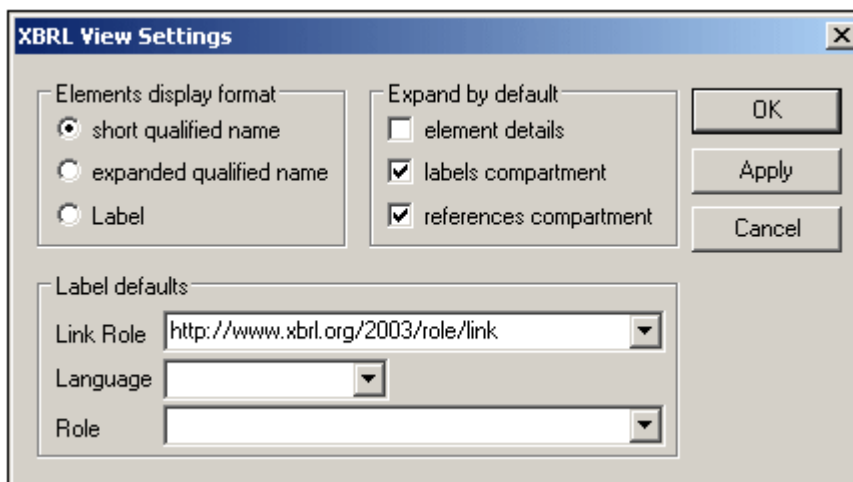
Clicking the **Edit** button in the Documentation Design pane launches StyleVision and opens the selected SPS in a StyleVision window. In order to preview the result document in StyleVision, you will need a Working XML file. A sample XML file for this purpose, called `nanonull.xml`, is supplied with your application and is located in the folder:

`C:\Documents and Settings\<username>\My Documents\Altova\XMLSpy2012\Documentation\XBRL\SampleData`

Note: In order to use an SPS to generate XBRL taxonomy documentation, you must have StyleVision installed on your machine.

21.15.7 View Settings

The **View Settings** command pops up the XBRL View Settings dialog (*screenshot below*), in which you can specify default settings for XBRL View.



The following settings can be made:

- **Display of element names** can be set to the short or expanded qualified name or to

labels. These defaults apply to the Main Window and the Details entry helper. The element-name display setting for the Global Elements entry helper is independent of the settings made in the XBRL View Settings dialog; the settings are in the [entry helper's menu bar](#).

- **Expand by default:** In the Main Window, element details, the labels box, and the references box can be set by default to the expanded state. Note that, if the labels or references boxes are set to be shown expanded by default, then the expanded boxes will be visible only when the Element details are expanded (either by default or manually). These default settings are visible when the view is refreshed.
- **Label defaults** for the label link role, the label language and the kind of label (link role) can be specified. The combo box for each property displays a list of available values.

21.15.8 Generate XBRL from DB, Excel, CSV with MapForce

This command starts the generation of an XBRL instance file based on the currently active taxonomy. The data for the instance file is obtained from an MS Excel datasheet, a database, or a CSV file. The XBRL instance file is generated by [Altova's MapForce program](#) which you must have installed on your machine. The command starts Altova MapForce and loads the taxonomy into it. You can then specify the source data file and graphically design the required instance file output.

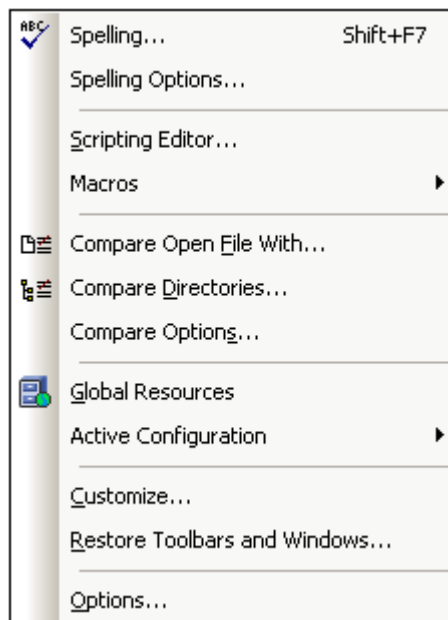
21.15.9 Present XBRL as HTML/PDF/Word with StyleVision

This command loads the currently active taxonomy into [Altova StyleVision](#), which enables you to generate a design for reports based on the taxonomy. In order to use this command, you must have Altova's StyleVision program installed on your machine.

21.16 Tools Menu

The tools menu allows you to:

- Check the [spelling](#) of your XML documents
- Access the [scripting environment](#) of XMLSpy. You can create, manage and store your own forms, macros and event handlers
- [View](#) the currently assigned macros
- Compare any two files to check for differences
- Compare any two folders to check for differences
- [Define global resources](#)
- [Change the active configuration](#) for global resources in XMLSpy
- [Customize](#) your version of XMLSpy: define your own toolbars, keyboard shortcuts, menus, and macros
- Define global XMLSpy [settings](#)



21.16.1 Spelling

XMLSpy's spellchecker with built-in language dictionaries (*see note below*) is enabled in Text View, Grid View, and Authentic View. If you wish to spellcheck a document that you have been editing in another view, you can switch to Text View or Grid View and run a spelling check. For example, if you have been editing an XML Schema document in Schema View, switch to Text View or Grid View for the spelling check.

Note: The selection of built-in dictionaries that ship with Altova software does not constitute any language preferences by Altova, but is largely based on the availability of dictionaries that permit redistribution with commercial software, such as the [MPL](#), [LGPL](#), or [BSD](#) licenses. Many other open-source dictionaries exist, but are distributed under more restrictive licenses, such as the [GPL](#) license. Many of these dictionaries are available as part of a separate installer located at <http://www.altova.com/dictionaries>. It is your choice as to whether you can agree to the terms of the license applicable to the dictionary and whether the dictionary is appropriate for your use with the software on your computer.

This section describes how to use the spellchecker. It is organized into the following

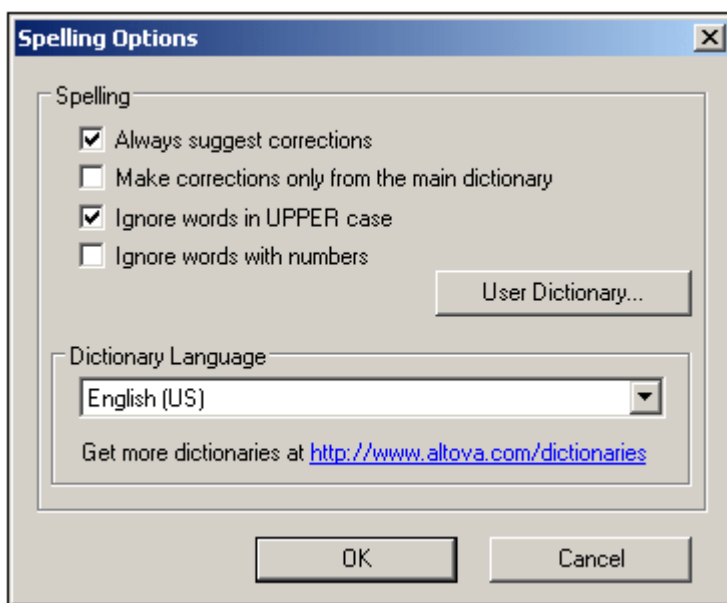
sub-sections:

- [Selecting the spellchecker language](#)
- [Defining the scope of the check](#)
- [Running the spelling check](#)

Selecting the spellchecker language

The spellchecker language can be set as follows:

1. Click the **Tools | Spelling Options** menu command.
2. In the Spelling Context dialog that pops up, click the **Spelling Options** button.
3. In the Spelling Options dialog (*screenshot below*), select one of the installed dictionaries from the dropdown list of the Dictionary Language combo box.

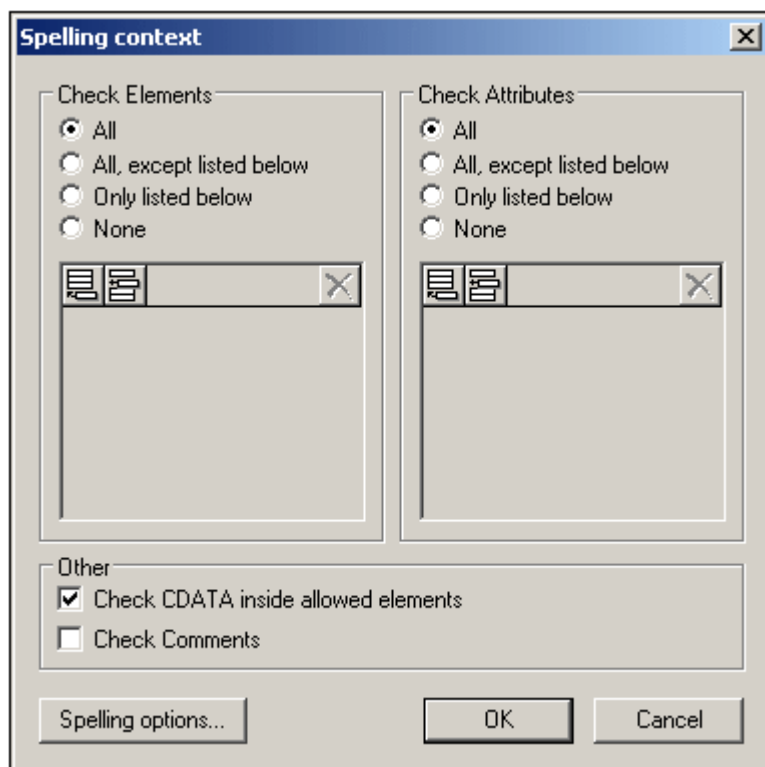


4. Click **OK** to finish.

The dictionary language you selected will be used by the spellchecker for spelling checks. If the language you want is not already installed, you can download additional language dictionaries. How to do this is described in the section, [Adding dictionaries for the spellchecker](#).

Defining the scope of the check

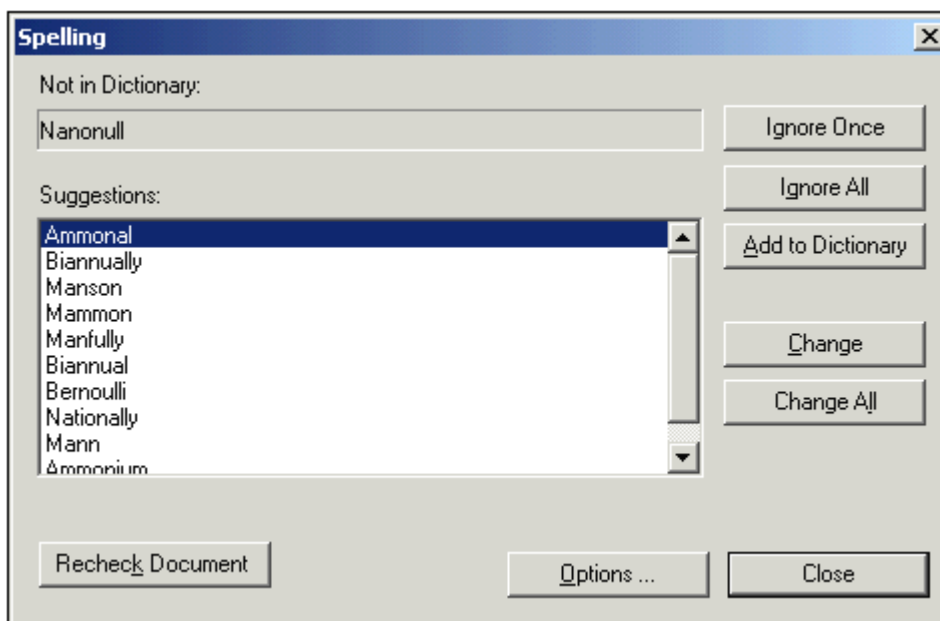
When the spellchecker is run in Text View or Grid View, the scope of the check can be defined immediately before starting the check. Do this by selecting the command **Tools | Spelling Options** and by defining the required scope in the [Spelling Context dialog](#) that pops up (see *screenshot below*).



You can select which elements and attributes are to be checked and whether CDATA sections and comments should be checked. Also see the description of the [Spelling Options](#) command for related information.

Running the spellchecker

The **Tools | Spelling (Shift+F7)** command automatically starts checking the currently active XML document according to the [defined scope](#). If an unknown word is encountered, the *Spelling: Not in Dictionary* dialog pops up (*screenshot below*). Otherwise the spelling check runs through to completion.



The various parts of the *Spelling: Not in Dictionary* dialog and the available options are described below:

Not in Dictionary

This text box contains the word that cannot be found in either the selected language dictionary or user dictionary. The following options are available:

- You can edit the word in the text box manually or select a suggestion from the *Suggestions* pane. Then click **Change** to replace the word in the XML document with the edited word. (Double-clicking a suggestion inserts it directly in the XML document.) When a word is shown in the *Not in Dictionary* text box, it is also highlighted in the XML document, so you can edit the word directly in the document if you like. Clicking **Change All** will replace all occurrences of the word in the XML document with the edited word.
- You can choose to not make any change and to ignore the spellchecker warning—either just for the current occurrence of the word or for every occurrence of it.
- You can add the word to the user dictionary and so allow the word to be considered correct for all checks from the current check onwards.

Suggestions

This list box displays words resembling the unknown word (supplied from the language and user dictionaries). Double-clicking a word in this list automatically inserts it in the document and continues the spellchecking process.

Ignore once

This command allows you to continue checking the document while ignoring the first occurrence of the unknown word. The same word will be flagged again if it appears in the document.

Ignore all

This command ignores all instances of the unknown word in the whole document.

Add to dictionary

This command adds the unknown word to the **user dictionary**. You can access the user dictionary (in order to edit it) via the [Spelling Options](#) dialog.

Change

This command replaces the currently highlighted word in the XML document with the (edited) word in the *Not in Dictionary* text box.

Change all

This command replaces all occurrences of the currently highlighted word in the XML document with the (edited) word in the *Not in Dictionary* text box.

Recheck Document

The **Recheck Document** button restarts the check from the beginning of the document.

Options

This command opens a dialog box depending on the current view.

- If the current view is Authentic View, the [Spelling Options](#) dialog box is opened.
- If the current view is Text View or Grid View, then the [Spelling Context](#) dialog box is opened.

For more information about these dialog boxes, see the section [Spelling Options](#).

Close

This command closes the Spelling dialog box.

21.16.2 Spelling Options

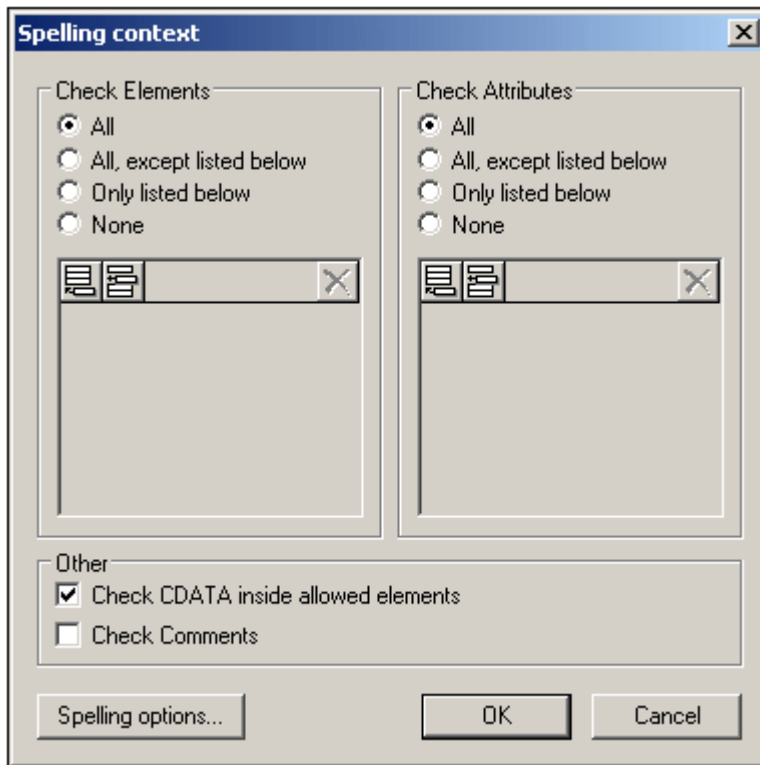
The **Tools | Spelling Options** command opens the [Spelling Options](#). Depending on which view is active, the **Tools | Spelling Options** command opens either the [Spelling Options](#) dialog directly (Schema View, WSDL View, XBRL View, Authentic View, Browser View), or the [Spelling Context](#) dialog (Text View, Grid View,). The Spelling Context dialog has a **Spelling Options** button to access the Spelling Options dialog.

The various settings available in these two dialogs are described in the sub-sections of this section:

- [Spelling context](#)
- [Spelling options](#)
- [Adding dictionaries for the spellchecker](#)
- [Working with the user dictionary](#)

Spelling context

Clicking the **Spelling Options** command in Text View or Grid View opens the Spelling Context dialog (*screenshot below*), in which you can select the scope of the spelling check. You can select which elements and attributes are to be checked and whether CDATA sections and comments should be checked.

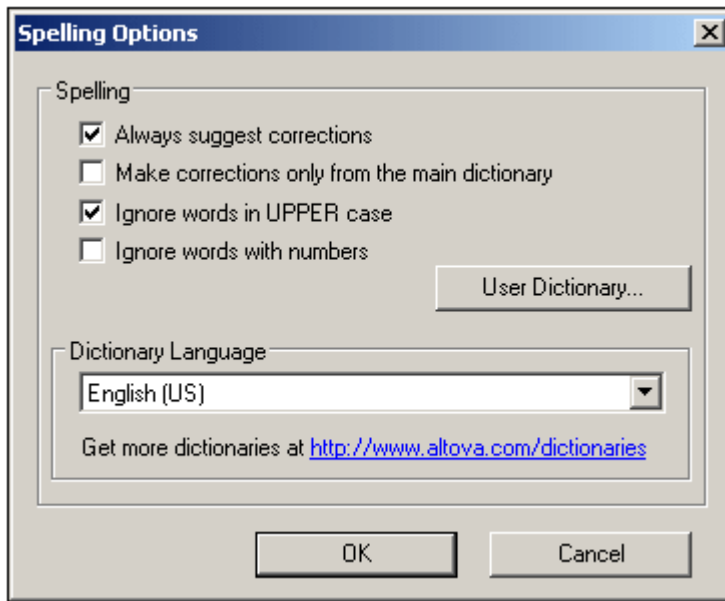


You can compile a list of elements and/or attributes that you wish to have spellchecked or have excluded from the spelling check. Alternatively, you can choose to run the spelling check on all elements and/or attributes or on no element or attribute.

Clicking the **Spelling Options** button at the bottom of the dialog opens the Spelling Options dialog.

Spelling options

The Spelling Options dialog is used to define global spellchecker options.



Always suggest corrections:

Activating this option causes suggestions (from both the language dictionary and the user dictionary) to be displayed in the Suggestions list box. Disabling this option causes no suggestions to be shown.

Make corrections only from main dictionary:

Activating this option causes only the language dictionary (main dictionary) to be used. The user dictionary is not scanned for suggestions. It also disables the **User Dictionary** button, preventing any editing of the user dictionary.

Ignore words in UPPER case:

Activating this option causes all upper case words to be ignored.

Ignore words with numbers:

Activating this option causes all words containing numbers to be ignored.

Dictionary Language

Use this combo box to select the dictionary language for the spellchecker. The default selection is US English. Other language dictionaries are available for download free of charge from the [Altova website](http://www.altova.com/dictionaries).

Adding dictionaries for the spellchecker

For each dictionary language there are two Hunspell dictionary files that work together: a .aff file and .dic file. All language dictionaries are installed in a `Lexicons` folder at the following location:

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:

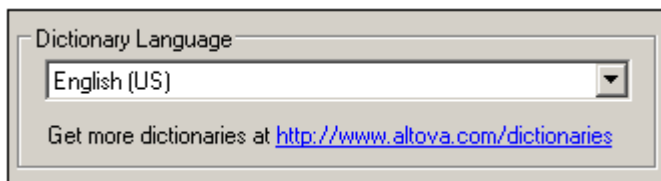
```
C:\ProgramData\Altova\SpellChecker\Lexicons\English (British)\en_GB.aff  
C:\ProgramData\Altova\SpellChecker\Lexicons\English (British)\en_GB.dic  
C:\ProgramData\Altova\SpellChecker\Lexicons\English (US)\en_US.dic  
C:\ProgramData\Altova\SpellChecker\Lexicons\English (US)\en_US.dic
```

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:

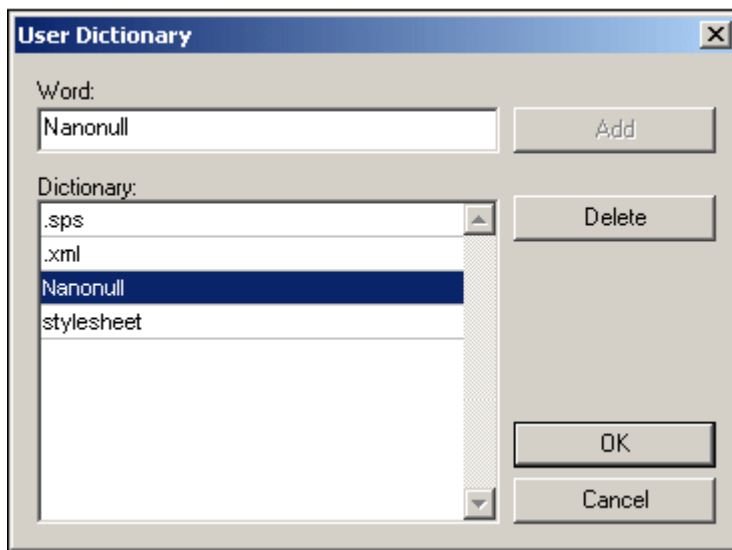
- By adding Hunspell dictionaries into a new subfolder of the `Lexicons` folder. Hunspell dictionaries can be downloaded, for example, from <http://wiki.services.openoffice.org/wiki/Dictionaryes> or <http://extensions.services.openoffice.org/en/dictionaries>. (Note that OpenOffice uses the zipped `oxt` format. So change the extension to `.zip` and unzip the `.aff` and `.dic` file to the language folders in the `Lexicons` folder. Also note that Hunspell dictionaries are based on Myspell dictionaries. So Myspell dictionaries can also be used.)
- By using the [Altova dictionary installer](#), which installs a package of multiple language dictionaries by default to the correct location on your machine. The installer can be downloaded 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*).



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

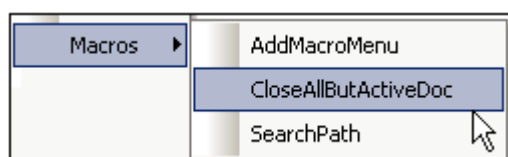
21.16.3 Scripting Editor

The Scripting Editor command opens the Scripting Editor window. How to work with the Scripting Editor is described in the [Scripting section](#) of this documentation.

Note: The .NET Framework version 2.0 or higher will have to be installed on your machine in order for the Scripting Editor to run.

21.16.4 Macros

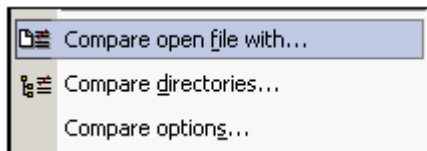
Mousing over the **Macros** command rolls out a submenu containing the macros defined in the Scripting Project that is currently active in XMLSpy (*screenshot below*).



Clicking a macro in the submenu (*see screenshot above*) runs the macro.

21.16.5 Comparisons

XMLSpy provides a comparison (or differencing) feature, with which you can compare XML and Text files, as well as folders, in order to check for differences.



There are the following menu items in the **Tools** menu that enable you to perform comparison tasks on files and folders:

- [Compare open file with](#)
- [Compare directories](#)
- [Compare options](#)

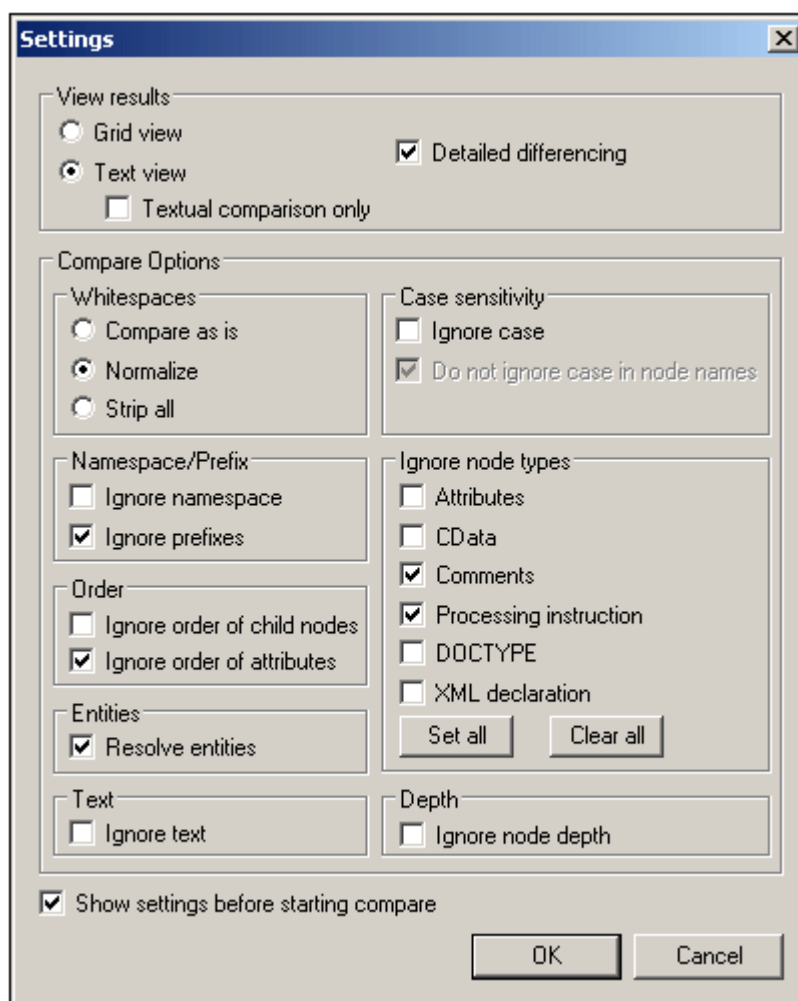
These commands are described in detail in the following sub-sections.

Compare Open File With

This command allows you to compare the open file with another file. The comparison shows both files tiled vertically in the main window with the differences between them highlighted in each file in green. You can compare files as XML documents (where the structure and semantics of tags is significant) or as text documents.

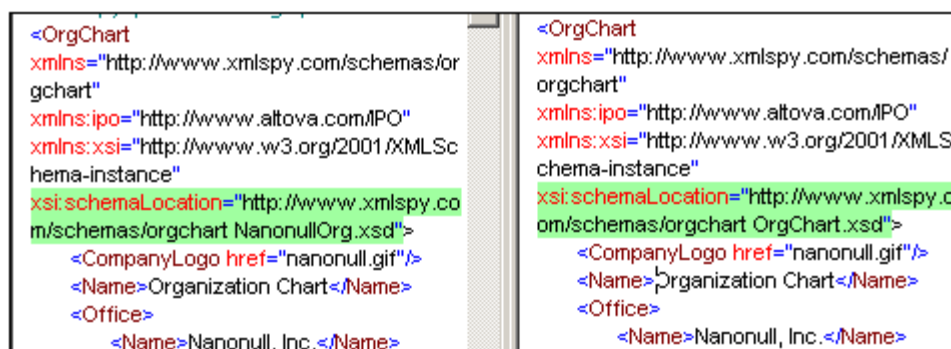
To compare the open file with another file:

1. With the file active in the Main Window (only one open file can be active in the Main Window at a given time), click **Tools | Compare open file with....**
2. Browse for the file you wish to compare the open file with, and select it. You can also select a file via a global resource or a URL (click the [Browse](#) button) or a file in one of the open windows in XMLSpy (click the **Window** button).
3. Click **OK**. The second file is opened, and the Settings dialog appears:

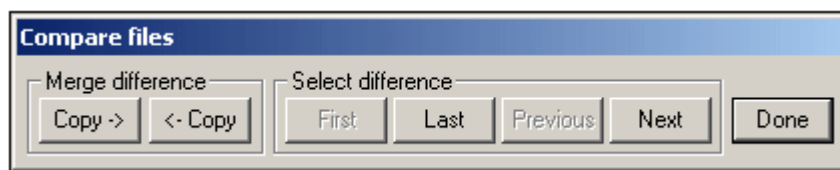


These settings are described under [Compare Options](#). If you do not wish to have this dialog appear each time you start a compare session, uncheck **Show settings before starting compare**, which is at the bottom of the dialog.

4. Select the required settings, then click **OK**. The two files now appear in the Main Window. Lines that are different are highlighted in green. One difference is selected at a time and is indicated in both files in a darker green color. (Also see the 'Highlight Colors' section below.)



A Compare Files control window also appears:



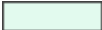
To select the next difference, click the **Next** button. The **First**, **Last**, and **Previous** buttons help you to navigate among the differences and to select a difference.

The "Merge difference" pane enables you to copy the selected difference from one file to the other. In order to enable merging, the following Compare options must be set: Detailed differencing must be checked and Ignore node depth must be unchecked. If you wish to undo a merge, stop the Compare session, select the file in which the change is to be undone, and select **Edit | Undo** or press **Ctrl + Z**.

5. When you are done with reviewing and/or merging differences, click **Done**.

Highlight colors

The differences and merged differences in the two files are indicated by the following highlight colors:

	<i>Difference</i>
	<i>Current difference</i>
	<i>Merged difference</i>
	<i>Current merged difference</i>

Please note:

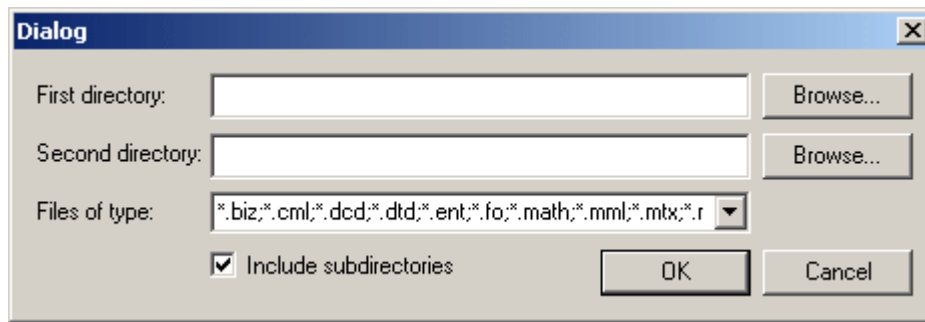
- While the Compare session is active, no editing or change of views is allowed. Any attempt to edit or change the view of either file will pop up a message warning that the Compare session will be ended.
- Compare settings can be changed during a Compare session (by selecting **Tools | Compare options...**), but will only take effect from the next Compare session onward; the new settings will not affect the current session.

Compare Directories

The Compare directories command allows you to compare two directories, with or without their sub-directories. Directories are compared to indicate missing files, and whether files of the same name are different or not.

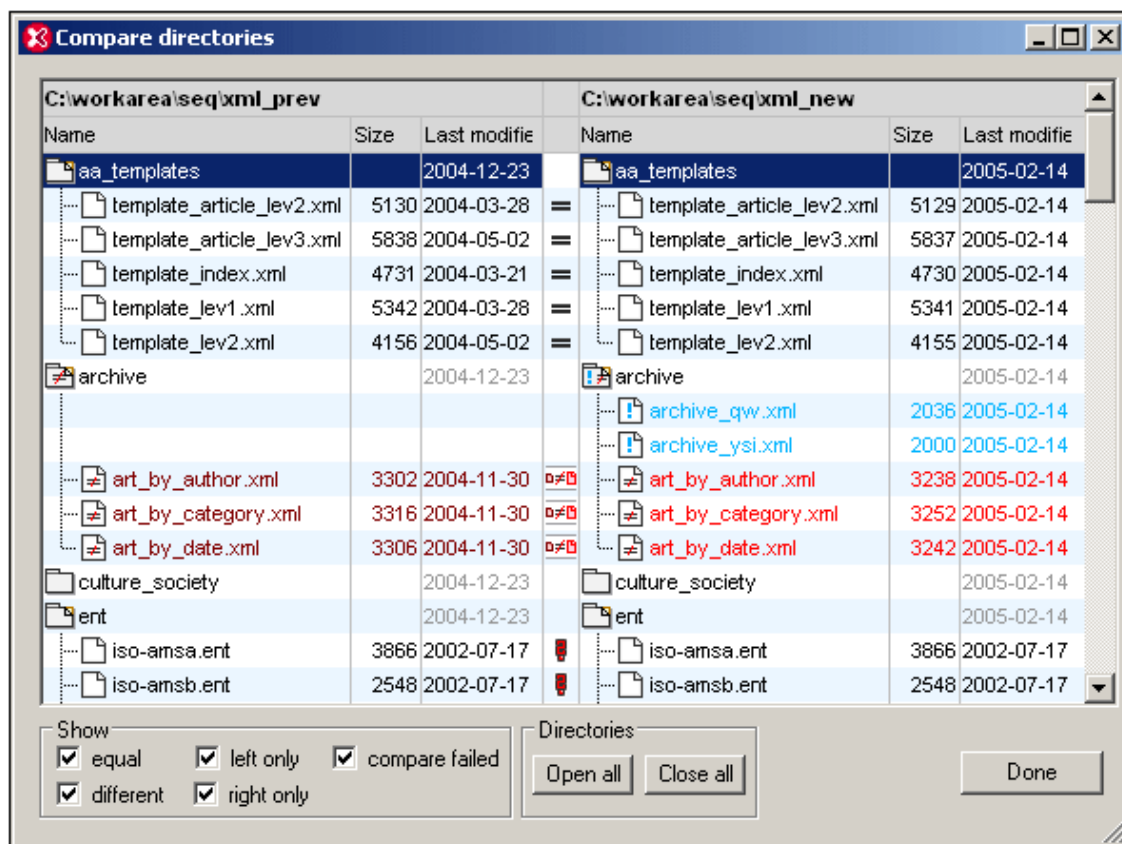
To compare two directories:

1. Click **Tools | Compare directories**. The following dialog appears.



2. Browse for the directories to compare, and check the **Include subdirectories** check box if you wish to include subdirectories in the Compare.
3. Select the file types you want to compare in the **Files of type** field. There are three options in the dropdown menu: (i) XML filetypes; (ii) [Filetypes defined in XMLSpy](#); and (iii) all filetypes. Zip archives are treated as directories, and directories and files in the Zip archive are displayed in the results window. To ensure that Zip archives are read as directories, make sure that the required extension is included in the Files of Type entry.
4. Click **OK**. The Settings dialog (described in [Compare Options](#)) appears.
5. Select the required settings for comparing files.
6. Click **OK**. A dialog will appear indicating the progress of the Compare.

The result will appear in a window, and will look like this:



Directory symbols

All directory names are given in black.

-  Directory is collapsed and its contents are not displayed.

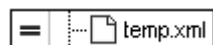
-  Directory is expanded, indicated by the turned down corner. The contents are displayed.
-  Directory contains files, all of which either cannot be compared or are not different from the corresponding file in the compared directory.
-  Directory contains one or more files that do not exist in the compared directory.
-  Directory contains one or more files that are different from the corresponding file in the compared directory.
-  Directory contains one or more files that do not exist in the compared directory and one or more files that are different from the corresponding file in the compared directory.

File symbols

The colors in which file names appear depend on their compare status. These colors are noted below.



This file cannot be compared (with the corresponding file in the compared directory). A question mark appears in the middle column. The file name appears in black.



This file is not different from the corresponding file in the compared directory. An equals-to sign appears in the middle column. The file name appears in black.



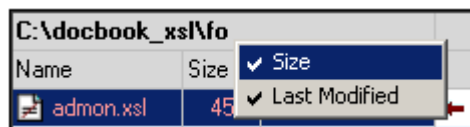
This file does not exist in the compared directory. The middle column is empty. The file name appears in blue.



This file is different from the corresponding file in the compared directory, and the last modification to this file is more recent than the last modification to the corresponding file. The newer file appears in a brighter red, and the icon shows the brighter red file symbol on the side having the newer file.

Viewing options

- Select what files to show by checking or unchecking the options in the Show pane at the bottom of the Result window.
- Open or close all subdirectories by clicking the appropriate button in the Directories pane.
- Expand or collapse subdirectories by double-clicking the folder icon.
- The Size and Last Modified can be toggled on and off by right-clicking on either file titlebar and clicking Size and Last Modified.



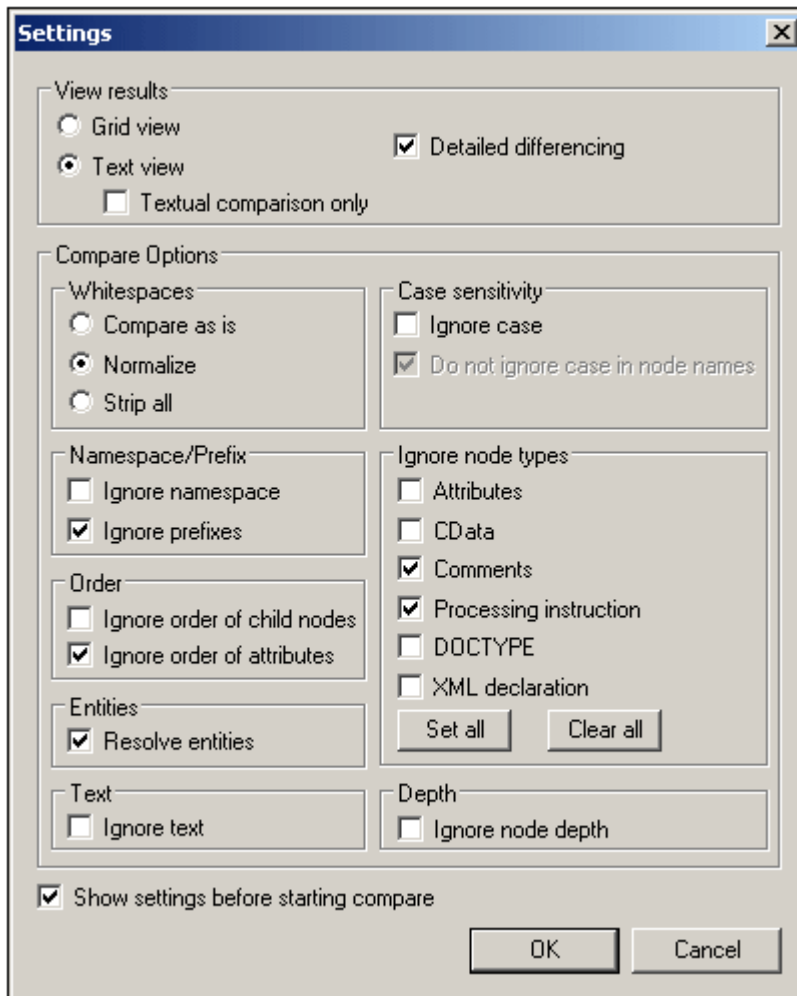
- Change column widths by dragging columns.
- The Result window can be maximized, minimized, and resized.

Comparing and merging files

Double-clicking on a line opens both files on that line in the Main Window, and directly starts a File Compare for the two files. You can then continue as in a regular Compare session (see [Compare open file with...](#)).

Compare Options

Click **Tools | Compare options** to open the Settings dialog (see *screenshot*). In it you make the settings for your Compare sessions. The settings that are current when a Compare session is started are the settings that are applied to that Compare session.



View results

Selects the view in which results are shown. You can select from the following options:

- Grid View (XML comparison)
- Text View with Textual Comparison Only unchecked (XML comparison)
- Text View with Textual Comparison Only checked (Text comparison)

If a view that provides XML comparison is selected, then the documents are treated as XML documents, and XML Compare Options are enabled. If Text comparison is selected, only Compare Options valid for Text comparison (Whitespaces and Case-sensitivity) are enabled; all other Compare Options are disabled.

Please note: You can merge differences in both Grid View and Text View, and in both XML and Text comparison modes. If you wish to undo a merge, stop the Compare session, select the file in which the change is to be undone, and select **Edit | Undo** or press **Ctrl + Z**.

Detailed differencing

If unselected, differences in immediate sibling elements are represented as a single difference, and the merge option is disabled. If selected, differences in immediate siblings are represented as separate differences, and merging is enabled.

Please note: The **Detailed differencing** check box must be checked to enable merging.

Whitespaces

Whitespace characters are space, tab, carriage return, and line feed. The options here compare files with whitespace unchanged; with whitespace normalized (i.e., all consecutive whitespace characters are reduced to one whitespace character); and with all whitespace stripped. The Whitespaces option is available for both XML and Text comparisons.

Case sensitivity

If the **Ignore case** check box is checked, then you have the option of ignoring or not ignoring case in node names (for XML comparisons only). The Case-sensitivity option is available for both XML and Text comparisons.

Namespace/Prefix

These are options for ignoring namespaces and prefixes when searching for differences.

Order

If **Ignore order of child nodes** is checked, then the position of child nodes relative to each other does not matter. The comparison is made for the entire set of child nodes, and if the only difference between a child node in one document and a child node in the compare document is the relative position in the nodeset, then this difference is ignored.

Please note: Each child element node is identified by its name, its attributes, and its position. If the names of more than one node in the sibling set are the same, then the position of these nodes is used to identify the nodes even if the "Ignore order of child nodes" option is checked. This option applies to each level separately.

If **Ignore order of child nodes** is unchecked, then differences in order are represented as differences.

The option of ignoring the order of attributes is also available, and applies to the order of attributes of a single element.

Entities

If "Resolve entities" is selected, then all entities in the document are resolved. Otherwise the files are compared with the entities as is.

Text

If "Ignore text" is selected, then differences in corresponding text nodes are not reported.

Ignore node types

Check the node types that will not be compared in the Compare session. Node types that may be ignored are Attributes, CDATA, Comments, Processing Instructions, DOCTYPE statements, and the XML declaration.

Depth

If **Ignore node depth** is checked, then the additional depth of any element (i.e. more levels of descendants) relative to the depth of the corresponding element in the compared file is ignored. This option must be unchecked to enable merging.

Show settings before starting compare

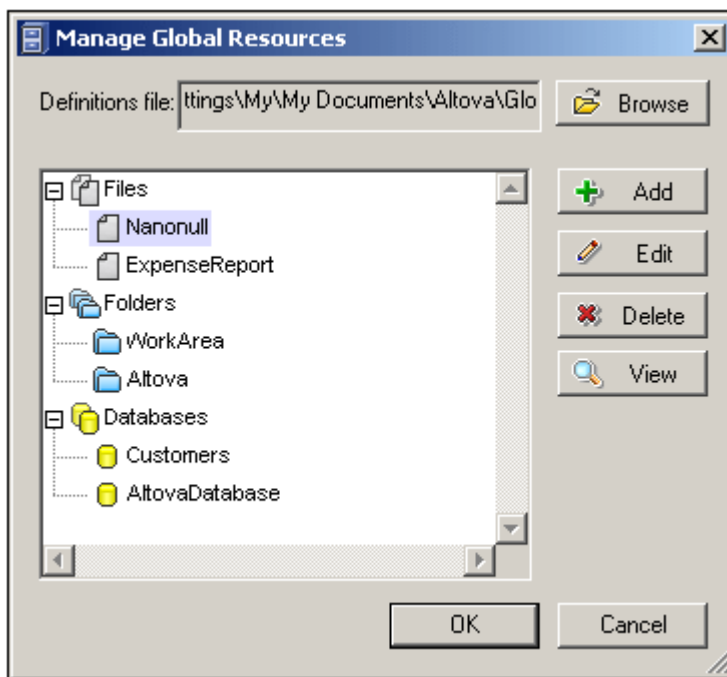
Checking this option causes this dialog to appear whenever the **Compare open file with** command is clicked. Having the Settings dialog appear before each comparison allows you to check and modify the settings for each comparison.

If this command is unchecked, then the Compare session will start directly when a comparison is invoked.

21.16.6 Global Resources

The **Global Resources** command pops up the Global Resources dialog (*screenshot below*), in which you can:

- Specify the Global Resources XML File to use for global resources.
- Add file, folder, and database global resources (or aliases)
- Specify various configurations for each global resource (alias). Each configuration maps to a specific resource.

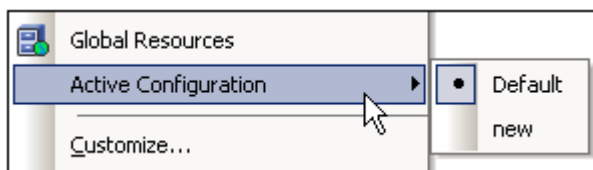


How to define global resources is described in detail in the section, [Defining Global Resources](#).

Note: The Altova Global Resources dialog can also be accessed via the [Global Resources toolbar](#) (**Tools | Customize | Toolbars | Global Resources**).

21.16.7 Active Configuration

Mousing over the **Active Configuration** menu item rolls out a submenu containing all the configurations defined in the currently active [Global Resources XML File](#) (*screenshot below*).



The currently active configuration is indicated with a bullet. In the screenshot above the

currently active configuration is `Default`. To change the active configuration, select the configuration you wish to make active.

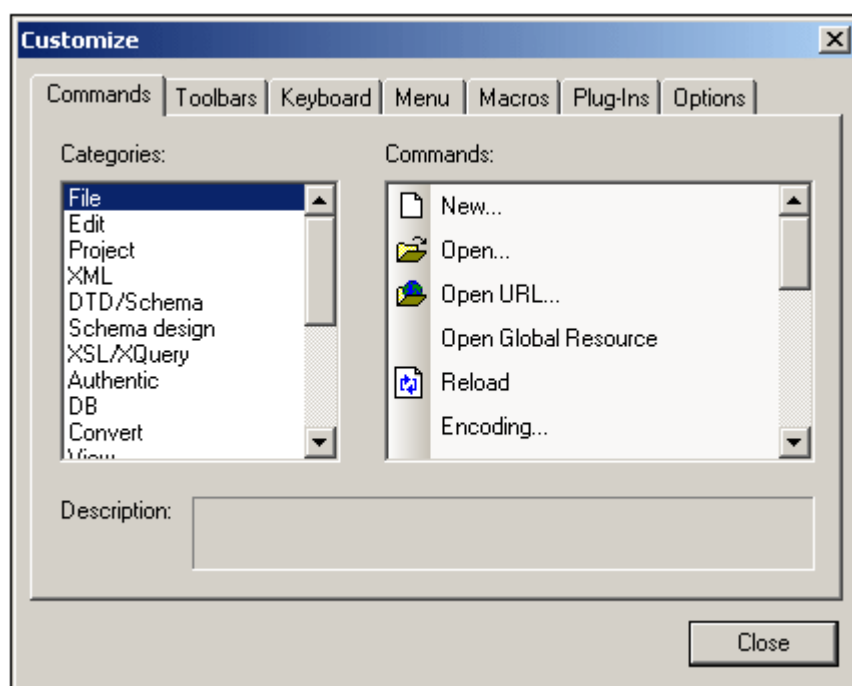
Note: The active configuration can also be selected via the [Global Resources toolbar](#) (**Tools | Customize | Toolbars | Global Resources**).

21.16.8 Customize

The **Customize** command lets you customize XMLSpy to suit your personal needs.

Commands

The **Commands** tab allows you customize your menus or toolbars.



To add a command to a toolbar or menu:

1. Select the menu item **Tools | Customize**. The Customize dialog appears.
 2. Select the **All Commands** category in the Categories list box. The available commands appear in the Commands list box.
 3. Click on a command in the Commands list box and drag it to an existing menu or toolbar. An I-beam appears when you place the cursor over a valid position to drop the command.
 4. Release the mouse button at the position you want to insert the command.
- A small button appears at the tip of mouse pointer when you drag a command. The "x" below the pointer means that the command cannot be dropped at the current cursor position.
 - The "x" disappears whenever you can drop the command (over a tool bar or menu).
 - Placing the cursor over a menu when dragging opens it, allowing you to insert the command anywhere in the menu.
 - Commands can be placed in menus or tool bars. If you created your own toolbar you can populate it with your own commands/icons.

Please note:

You can also edit the commands in the [context menus](#) (right-click anywhere to open the context menu), using the same method. Click the **Menu** tab and then select the specific context menu available in the Context Menus combo box.

To delete a command or menu:

1. Select the menu item **Tools | Customize**. The Customize dialog appears.
2. Click on the menu entry or icon you want to delete, and drag with the mouse.
3. Release the mouse button whenever the "x" icon appears below the mouse pointer.
The command, or menu item, is deleted from the menu or tool bar.

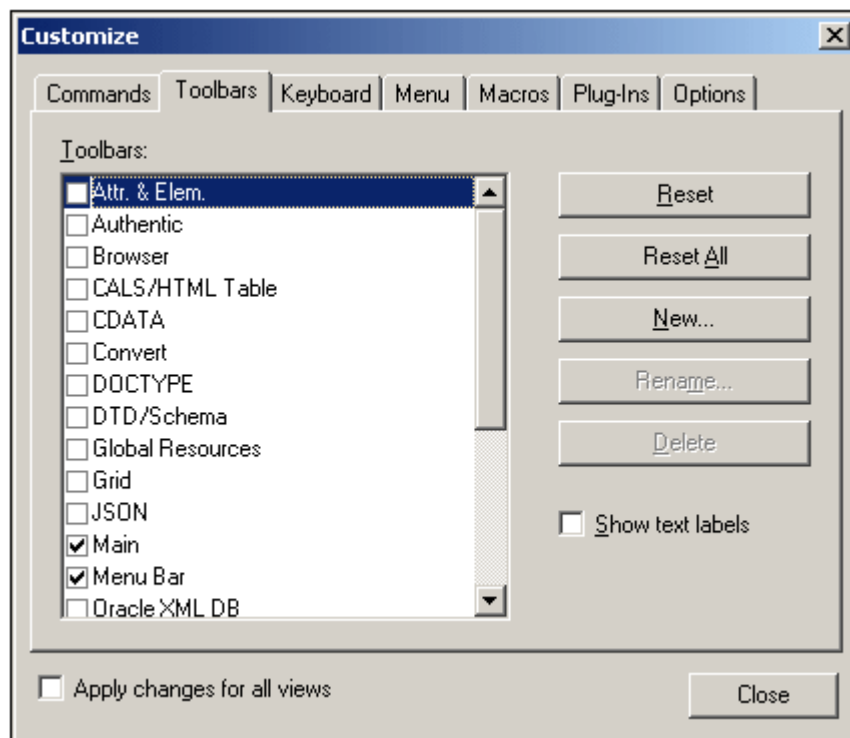
Toolbars

The **Toolbars** tab allows you to activate or deactivate specific toolbars, as well as create your own specialized ones.

XMLSpy toolbars contain symbols for the most frequently used menu commands. For each symbol you get a brief "tool tip" explanation when the mouse cursor is directly over the item and the status bar shows a more detailed description of the command.

You can drag the toolbars from their standard position to any location on the screen, where they appear as a floating window. Alternatively you can also dock them to the left or right edge of the main window.

- Toolbar settings defined in Grid View, Schema View, WSDL View and Text View are valid in those views. The Browser View toolbars are independent of all the other views.

**To activate or deactivate a toolbar**

- Click the check box to activate (or deactivate) the specific toolbar.

Apply changes for all views

- Check this check box at the bottom of the dialog to apply changes to all views (and not only to the current view). On clicking **Close**, all changes that were made **after** checking the check box will be applied to all views.

To create a new toolbar

1. Click the **New...** button, and give the toolbar a name in the Toolbar name dialog box.
2. Drag commands to the toolbar in the [Commands](#) tab of the Customize dialog box.

To reset the Menu Bar

1. Click the Menu Bar entry.
2. Click the **Reset** button, to reset the menu commands to the state they were in when XMLSpy was installed.

To reset all toolbar and menu commands

1. Click the **Reset All** button, to reset all the toolbar commands to the state they were when the program was installed. A prompt appears stating that all toolbars and menus will be reset.
2. Click **Yes** to confirm the reset.

To change a toolbar name

- Click the **Rename...** button to edit the name of the toolbar.

To delete a toolbar

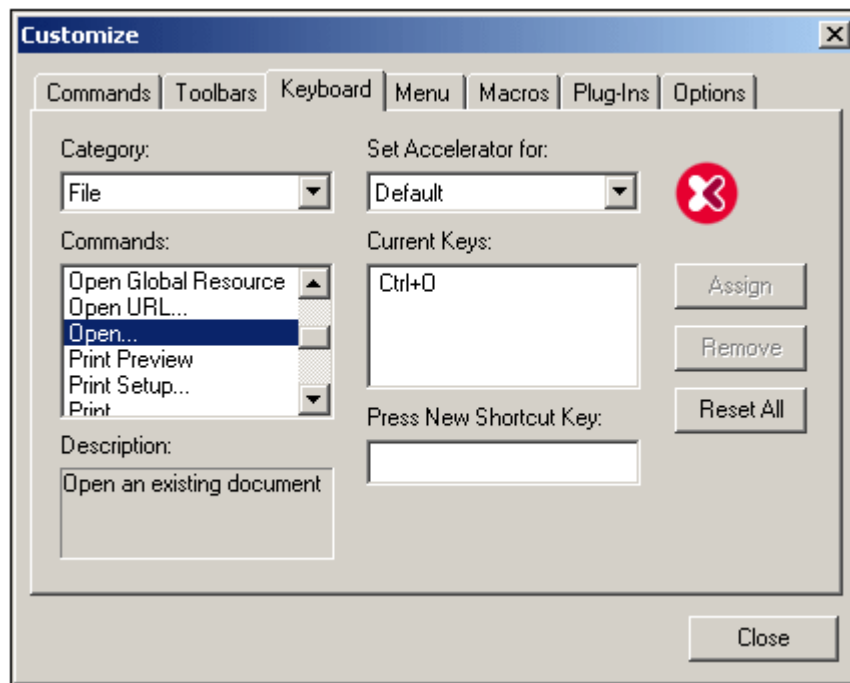
1. Select the toolbar you want to delete in the Toolbars list box.
2. Click the **Delete** button.
3. A prompt appears, asking if you really want to delete the toolbar. Click **Yes** to confirm the deletion.

Show text labels

This option displays explanatory text below toolbar icons when activated.

Keyboard

The **Keyboard** tab allows you to define (or change) keyboard shortcuts for any XMLSpy command.

**To assign a new Shortcut to a command:**

1. Select the All Commands category using the **Category** combo box. Note that if a [macro has been selected as an Associated Command](#), then macros can also be selected via the category combo box and a shortcut for the macro can be set.
2. Select the **command** you want to assign a new shortcut to, in the Commands list box
3. Click in the **Press New Shortcut Key:** text box, and press the shortcut keys that are to activate the command.
The shortcuts appear immediately in the text box. If the shortcut was assigned previously, then that function is displayed below the text box.
4. Click the **Assign** button to assign the shortcut.
The shortcut now appears in the Current Keys list box.
(To **clear** this text box, press any of the control keys, **CTRL**, **ALT** or **SHIFT**).

To de-assign or delete a shortcut:

1. Click the shortcut you want to delete in the Current Keys list box.
2. Click the **Remove** button.
3. Click the **Close** button to confirm.

Set accelerator for:

Currently no function.

Currently assigned keyboard shortcuts:**Hotkeys by key**

F1	Help Menu
F3	Find Next
F5	Refresh
F7	Check well-formedness
F8	Validate
F9	Insert/Remove breakpoint
Shift+F9	Insert/Remove tracepoint
CTRL+F9	Enable/Disable breakpoint
Shift+CTRL+F9	Enable/Disable tracepoint
F10	XSL Transformation
CTRL+F10	XSL:FO Transformation
F11	Step into
CTRL+F11	Step Over
Shift + F11	Step Out
Alt+F11	Start Debugger/Go
Num +	Expand
Num -	Collapse
Num *	Expand fully
CTRL+Num-	Collapse unselected
CTRL + G	Goto line/char
CTRL+TAB	Switches between open documents
CTRL+F6	Cycle through open windows
Arrow keys (up / down)	Move selection bar
Esc.	Abandon edits/close dialog box
Return/Space bar	confirms a selection
Alt + F4	Closes XMLSpy
CTRL + F4	Closes active window
Alt + F, 1	Open last file
CTRL + Double click an element (Schema view)	Display element definition
CTRL + N	File New
CTRL + O	File Open
CTRL + S	File Save
CTRL + P	File Print
CTRL + A	Select All
Shift + Del	Cut (or CTRL + X)
CTRL + C	Copy
CTRL + V	Paste
CTRL + Z	Undo
CTRL + Y	Redo
Del	Delete (Delete item in Schema/Grid View)
CTRL + F	Find
F3	Find Next

CTRL + H	Replace
CTRL + I	Append Attribute
CTRL + E	In Grid View, Append Element. in Text View, Jump to Start/End Tag when cursor is in other member of the pair
CTRL + T	Append Text
CTRL + D	Append CDATA
CTRL + M	Append Comment
CTRL + SHIFT + I	Insert Attribute
CTRL + SHIFT + E	Insert Element
CTRL + SHIFT + T	Insert Text content
CTRL + SHIFT + D	Insert CDATA
CTRL + SHIFT + M	Insert Comment
CTRL + ALT + I	Add Child Attribute
CTRL + ALT + E	Add Child Element
CTRL + ALT + T	Add Child Text
CTRL + ALT + D	Add Child CDATA
CTRL + ALT + M	Add Child Comment
Hotkeys for Text View	
CTRL + "+"	Zoom In
CTRL + "-"	Zoom Out
CTRL + 0	Reset Zoom
CTRL + mouse wheel forward	Zoom In
CTRL + mouse wheel back	Zoom Out

Currently assigned keyboard shortcuts:**Hotkeys by function**

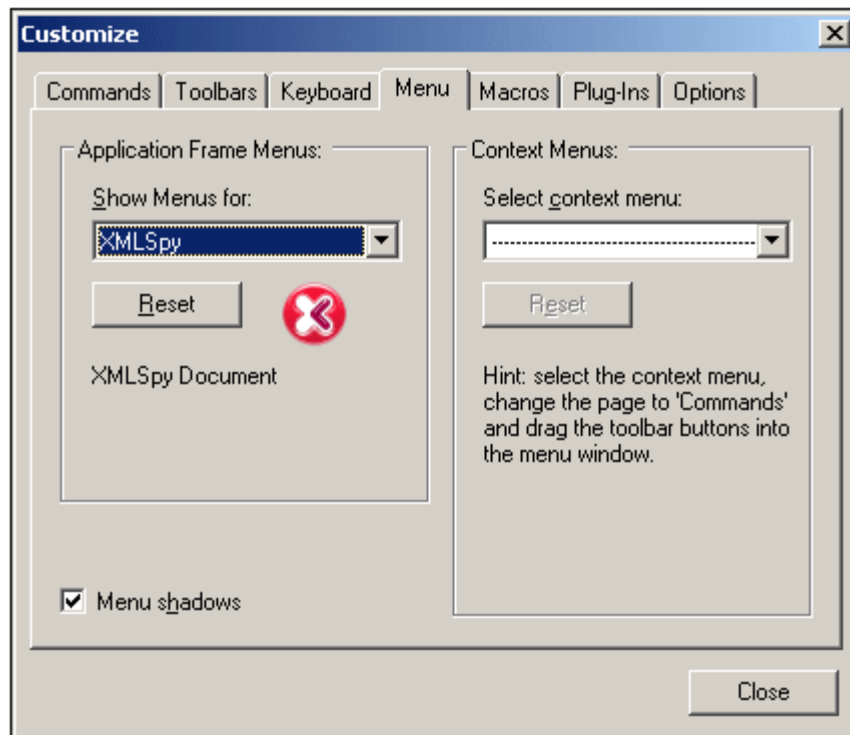
Abandon edits	Esc.
Add Child Attribute	CTRL + ALT + I
Add Child CDATA	CTRL + ALT + D
Add Child Comment	CTRL + ALT + M
Add Child Element	CTRL + ALT + E
Add Child Text	CTRL + ALT + T
Append Attribute	CTRL + I
Append CDATA	CTRL + D
Append Comment	CTRL + M
Append Element	CTRL + E (Grid View)
Append Text	CTRL + T
Check well-formedness	F7
Closes active window	CTRL + F4
Close XMLSpy	Alt + F4
Collapse	Num -
Collapse unselected	CTRL + Num-
Confirms a selection	Return / Space bar
Copy	CTRL + C
Cut	SHIFT + Del (or CTRL + X)
Cycle through windows	CTRL + TAB and CTRL + F6
Delete item	Del
Enable/Disable breakpoint	CTRL + F9
Enable/Disable tracepoint	Shift + CTRL + F9
Expand	Num +
Expand fully	Num *
File New	CTRL + N
File Open	CTRL + O
File Print	CTRL + P
File Save	CTRL + S
Find	CTRL + F
Find Next	F3
Goto line/char	CTRL + G
Help Menu	F1
Highlight other tag in pair when cursor is inside a start or end element tag	CTRL + E (Text View)
Insert Attribute	CTRL + SHIFT + I
Insert CDATA	CTRL + SHIFT + D
Insert Comment	CTRL + SHIFT + M
Insert Element	CTRL + SHIFT + E
Insert/Remove breakpoint	F9
Insert/Remove tracepoint	SHIFT+F9
Insert Text content	CTRL + SHIFT + T
Move selection bar	Arrow keys (up / down)
Open last file	Alt + F, 1
Paste	CTRL + V
Redo	CTRL + Y
Refresh	F5
Replace	CTRL + H
Select All	CTRL + A
Start Debugger/Go	Alt + F11

Step Into	F11
Step Out	Shift + F11
Step Over	CTRL + F11
To view an element definition	CTRL + Double click on an element.
Undo	CTRL + Z
Validate	F8
XSL Transformation	F10
XSL:FO Transformation	CTRL + F10

In the application, you can see a list of commands, together with their shortcuts and descriptions, in the Keyboard Map dialog ([Help | Keyboard Map](#)).

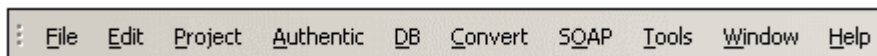
Menu

The **Menu** tab allows you to customize the main menu bars as well as the (popup - right click) context menus.



You can customize both the Default and XMLSpy menu bars.

The **Default** menu (*screenshot below*) is the one visible when no XML documents of any type are open in XMLSpy.



The XMLSpy menu is the menu bar visible when at least one XML document has been opened.

To customize a menu:

1. Select the menu bar you want to customize from the **Show Menus for:** combo box
2. Click the [Commands](#) tab, and drag the commands to the menu bar of your choice.

To delete commands from a menu:

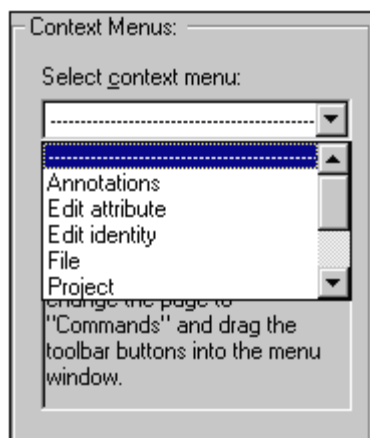
1. Click right on the command, or icon representing the command.
 2. Select the **Delete** option from the popup menu,
- or,
1. Select **Tools | Customize** to open the Customize dialog box.
 2. Drag the command away from the menu, and drop it as soon as the check mark icon appears below the mouse pointer.

To reset either of the menu bars:

1. Select either the Default or XMLSpy entry in the **Show Menus for** 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 any of the Context menus (right-click menus):

1. Select the context menu from the **Select context menu** combo box. The context menu you selected appears.
2. Click the [Commands](#) tab, and drag the commands to the context menu.

**To delete commands from a context menu:**

1. Click right on the command, or icon representing the command.
 2. Select the **Delete** option from the popup menu
- or,
1. Select **Tools | Customize** to open the Customize dialog box.
 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 any of the context menus:

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

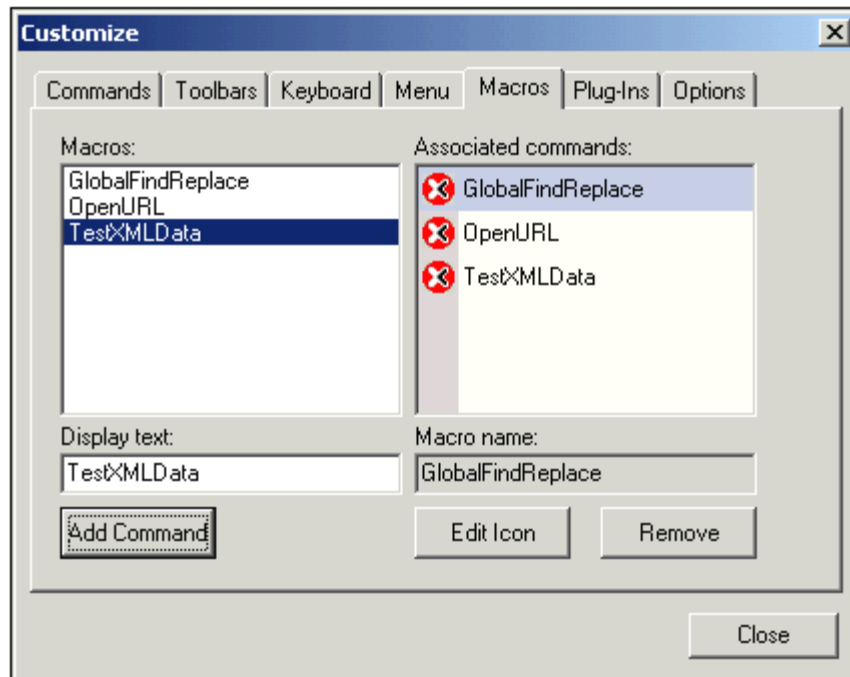
- Click the Close button of the Customize dialog box.

Menu shadows

- Click the **Menu shadows** check box, if you want all your menus to have shadows.

Macros

The **Macros** tab allows you to place macros (created using the XML scripting environment) in a toolbar or menu.



To place a macro (icon) into a toolbar or menu:

1. Start the scripting environment using **Tools | Switch to Scripting environment**.
2. Double-click the XMLSpyMacros entry in the Modules folder of the **XMLSpyGlobalScripts** project.
Previously defined macros will then be visible in the right hand window.
3. Switch back to XMLSpy, and select **Tools | Customize** and click on the **Macros** tab.
The macros defined in the scripting environment are now visible in the Macros list box at left.
4. Click the macro name and then the **Add Command** button. This places the macro name in the Associated Commands list box.
5. Click the macro name in the Associated Commands list box, and drag it to any tool bar or menu.

To edit a macro icon:

- Click the **Edit Icon** button.

To delete a macro from the Associated commands list box:

- Click the **Remove** button.

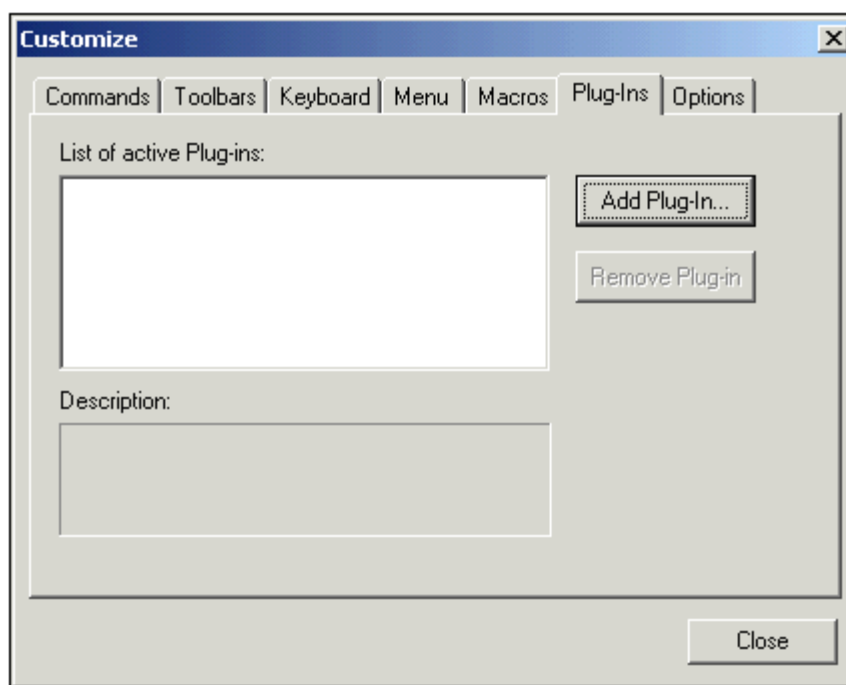
Note: If a macro has been set as an Associated Command, you can set a [keyboard shortcut for it](#). In the Keyboard tab of the Customize dialog, select Macros in the Category combo box, then select the required macro, and set the shortcut. You should set a macro as an associated command in order for Macros to be available for keyboard shortcuts.

Plug-Ins

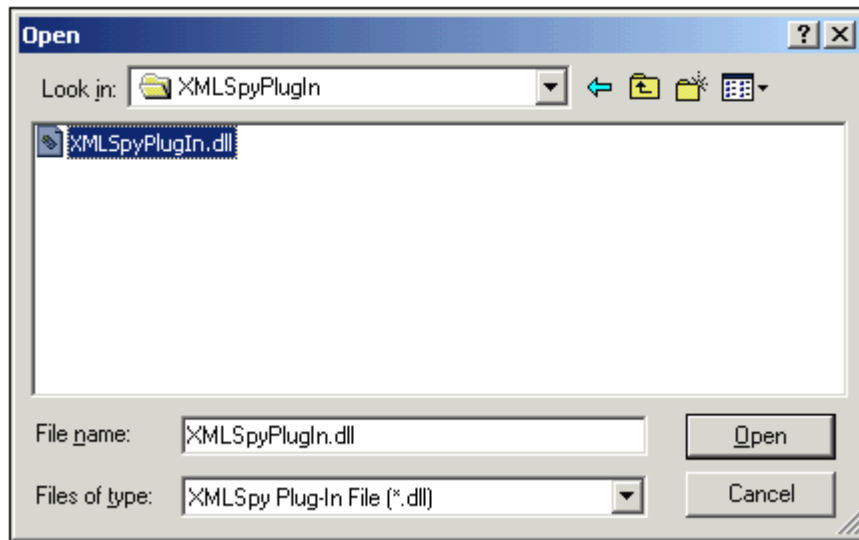
The Plug-Ins tab allows you to place plug-ins in a toolbar or menu.

To place a plug-in icon into a toolbar or menu:

1. Click the **Add Plug-In...** button.



2. Select the folder and then click the plug-in file to mark it (XMLSpyPlugIn. dll in this case).



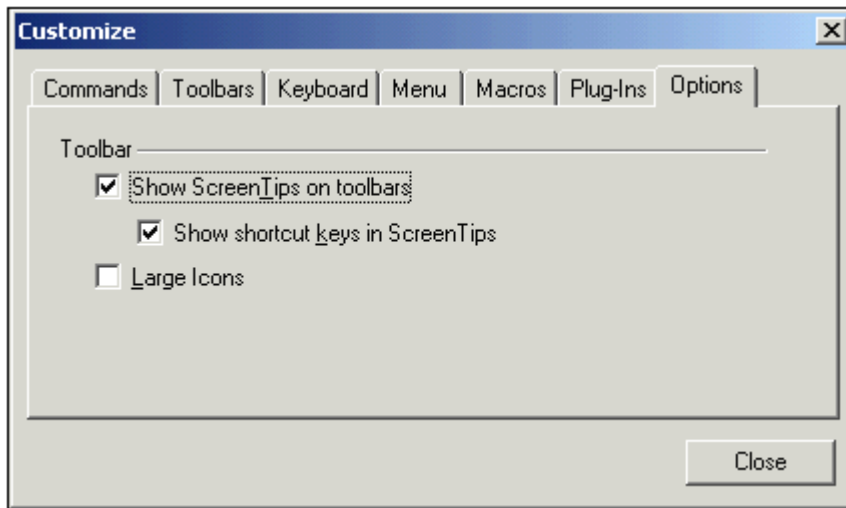
3. Click the **Open** button to install the plug-in.
The plug-in name appears in the "list of active plug-ins" list, and the plug-in icon(s) appears in a new toolbar.

To remove a plug-in:

- Click the plug-in name in the "List of active plug-ins" list and click the "Remove Plug-in" button

Options

The Options tab allows you to set general environment settings.



Toolbar

When active, the **Show ScreenTips on toolbars** check box displays a popup when the mouse pointer is placed over an icon in any of the icon bars. The popup contains a short description of the icon function, as well as the associated keyboard shortcut, if one has been assigned.

The **Show shortcut keys in ScreenTips** check box, allows you to decide if you want to have the shortcut displayed in the tooltip.

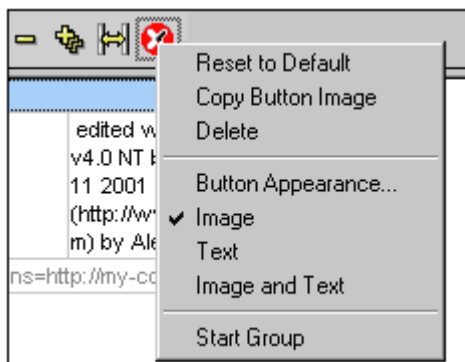
When active, the **Large icons** check box switches between the standard size icons, and larger versions of the icons.

Customize Context Menu

The Customize context menu allows you to further customize icons and menu items.

To open the customize context menu:

1. First open the Customize dialog by selecting **Tools | Customize**.
2. Then place the mouse pointer over an icon (or menu) and **click right** to open the context menu.



Reset to default

Currently no function.

Copy Button image

This option copies the icon you right-click to the clipboard.

Delete

This option deletes the icon or menu you right click. Deleting a menu deletes all menu options contained in it!

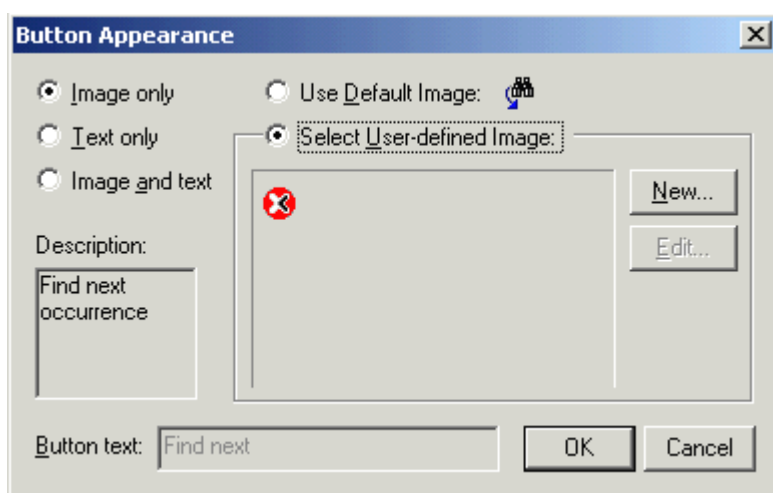
To restore a deleted menu:

1. Select the **Menu** tab in the Customize dialog.
2. Select the menu you want to restore (XMLSpy or Default).
3. Click the **Reset** button below the menu selection combo box.

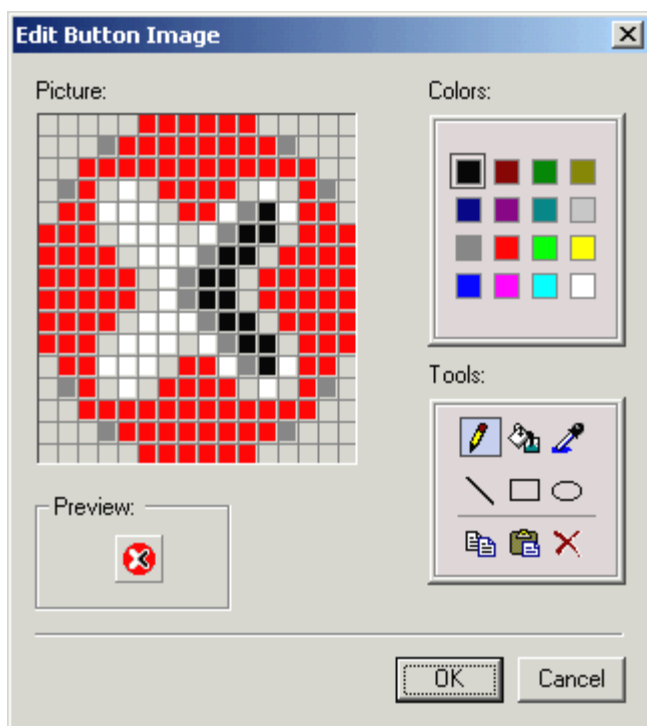
Button Appearance...

This option allows you to edit the button image as well as the button text. Currently only the macro icons can be edited (default XMLSpy icon).

The Image only, Text only and Image and text radio buttons, let you define what you want to edit. Click the Select User-defined Image radio button and click one of the icons.



Click the **Edit...** button, to open the Edit button image dialog box. The Button text can be edited if you select either **Text only** or **Image and text** options.



Image

This option changes the textual description of a function to the graphical image (icon) representing that function.

Text

This option changes the graphical display of an icon to the text representing it.

Image and Text

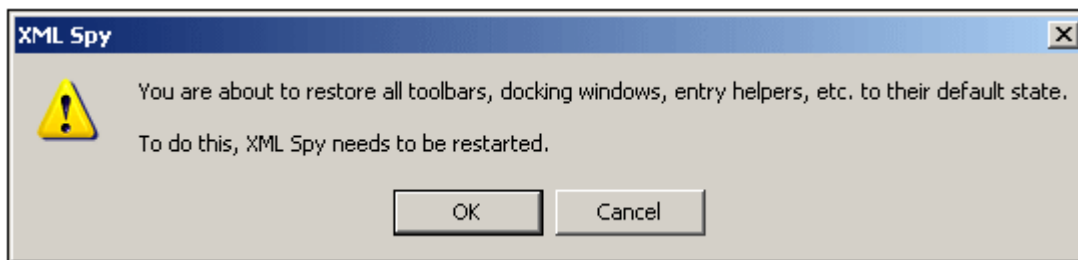
This option enables the simultaneous display of an icon and its text.

Start group

This option inserts a vertical divider to the left of the icon you right clicked.

21.16.9 Restore Toolbars and Windows

The **Restore Toolbars and Windows** command closes down XMLSpy and re-starts it with the default settings. Before it closes down a dialog pops up asking for confirmation about whether XMLSpy should be closed (*screenshot below*).



This command is useful if you have been resizing, moving, or hiding toolbars or windows, and would now like to have all the toolbars and windows as they originally were.

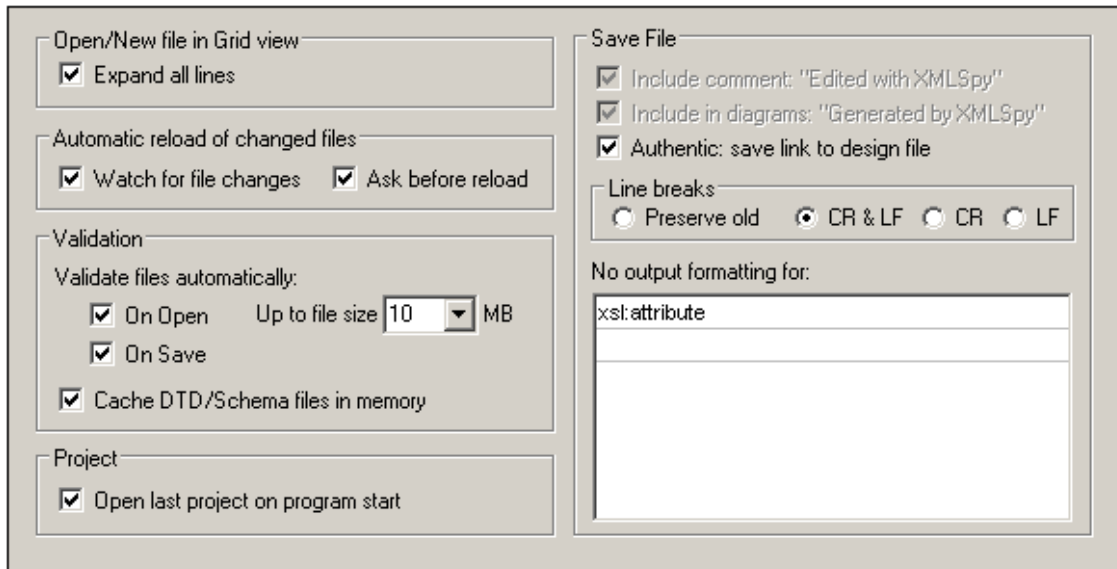
21.16.10 Options

The **Tools | Options** command enables you to define global application settings. These settings are specified in a tabbed dialog box and saved in the registry. They apply to all current and future document windows. The **Apply** button in the Options dialog displays the changes in the currently open documents and fixes the current settings. The changes are seen immediately in the background windows.

Each tab of the Options dialog is described in detail in this section.

File

The **File** tab defines the way XMLSpy opens and saves documents. Related settings are in the [Encoding tab](#).



Open/New file in Grid view

You can choose to open an existing file or create a new file either in Grid View or in Text View. If you select Grid View, you can also choose to automatically expand all lines.

Automatic reload of changed files

If you are working in a multi-user environment, or if you are working on files that are dynamically generated on a server, you can watch for changes to files that are currently open in the interface. Each time XMLSpy detects a change in an open document, it will prompt you about whether you want to reload the changed file.

Validation

If you are using DTDs or schemas to define the structure of your XML documents, you can automatically check the document for validity whenever it is opened or saved. During Open and Save operations, you have the option of validating files only if the file-size is less than a size you specify in MB. If the document is not valid, an error message will be displayed. If it is valid, no message will be displayed and the operation will proceed without any notification. XMLSpy can also cache these files in memory to save any unnecessary reloading (e.g. when the schema being referred to is accessed through a URL). If your schema location declaration uses an URL, disable the "cache DTD/Schema files in memory" option to have changes made to the schema appear immediately, and not use the cached version of the schema.

Project

When you start XMLSpy, you can open the last-used project automatically.

Save File

When saving an XML document, XMLSpy includes a short comment `<!-- Edited with XMLSpy http://www.altova.com -->` near the top of the file. This option can only be deactivated by licensed users, and takes effect when editing or saving files in the Enhanced Grid or Schema Design View.

When saving a content model diagram (using the menu option **Schema design | Generate Documentation**), XMLSpy includes the XMLSpy logo. This option can only be deactivated by licensed users.

If a StyleVision Power Stylesheet is associated with an XML file, the 'Authentic: save link to design file' option will cause the link to the StyleVision Power Stylesheet to be saved with the XML file.

Line breaks

When you open a file, the character coding for line breaks in it are preserved if **Preserve old** is selected. Alternatively, you can choose to code line breaks in any of three codings: **CR&LF** (for PC), **CR** (for MacOS), or **LF** (for Unix).

No output formatting for

In Text View, the indentation of an element can be made to reflect its position in the element hierarchy (see **Save File**). You can, however, override this indentation for individual elements. To do this, enter the element name in the **No output formatting for** field. All elements entered in this field will be formatted such that their descendant elements have no whitespace between them (see *screenshots*).

Hierarchical indentation for all elements:



No output formatting has been specified for element `xs:restriction`:

```

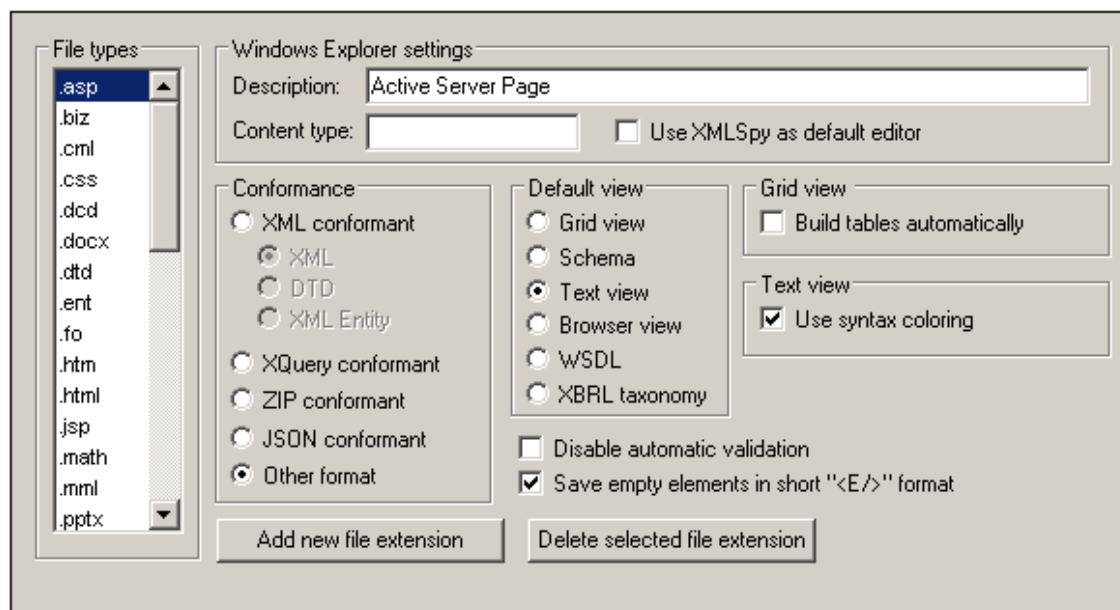
11  <xs:simpleType>
12      <xs:restriction base="xs:string"><xs:maxLength value="255"/></xs:restriction>
13  </xs:simpleType>

```

After making the settings, click **OK** to finish and close the Options dialog.

File Types

The **File types** tab allows you to customize the behavior of XMLSpy on a per-file-type basis.



Choose a file type from the File Types list box to customize the functions for that particular file type:

Windows Explorer settings

You can define the file type description and MIME-compliant content type used by Windows Explorer and whether XMLSpy is to be the default editor for documents of this file type.

Conformance

XMLSpy provides specific editing and other features for various file types. The features for a file type are set by specifying the conformance in this option. XMLSpy lets you set file type to conform with XML, XQuery, ZIP, JSON, and other (text) grammars. Furthermore, XML conformance is differentiated between XML, DTD, and XML Entity file types. A large number of file types are defined with a default conformance that is appropriate for the file type. We recommend that you do not modify these settings unless you are adding a new file type or deliberately wish to set a file type to another kind of conformance.

Default view

This group lets you define the default view to be used for each file type.

Grid View

This check box lets you define whether the Grid View should automatically build tables.

Text View

This check box lets you set syntax-coloring for particular file types.

Disable automatic validation

This option enables you to disable automatic validation per file type. Automatic validation typically takes place when a file is opened or saved, or when a view is changed.

Save empty elements in short <E/> format

Some applications that use XML documents or output generated from XML documents may have problems understanding the short `<Element/>` form for empty elements defined in the XML 1.0 Specification. You can instruct XMLSpy to save elements in the longer (but also valid) `<Element></Element>` form.

Add new file extension

Adds a new file type to the File types list. You must then define the settings for this new file type using the other options in this tab.

Delete selected file extension

Deletes the currently selected file type and all its associated settings.

After making the settings, click **OK** to finish and close the Options dialog.

Editing

The **Editing** tab enables you to specify editing behaviour in XMLSpy.

Intelligent editing

- ☒ Show entry helpers
 - ☒ Load entry helpers upon opening file
 - Sort: ☒ Attributes ☐ Elements
 - Mandatory first: ☒ Attributes ☐ Elements
- ☒ Autom. append mandatory children to new elements
 - ☐ Minimize number of elements
 - ☒ Generate mandatory elements only
 - ☐ Generate all elements
- ☐ Generate non-mandatory Attributes
- ☐ Treat element content of nillable elements as non-mandatory
- ☒ For elements with an abstract type, try to use a non-abstract type for xsi:type

Text View

- ☒ Auto-complete in Text View
- Disable auto-completion and entry helpers if file size is bigger than: MB

Table view

- ☒ Smart table detection for rep. elements
- ☐ Build table for any repeated elements
- ☐ Show single table subelements as table
- No automatic tables for elements:

p

Default copy to clipboard in Grid View as

- ☒ XML-Text
- ☐ Structured text (TAB-delimited)

Intelligent editing

While editing documents, XMLSpy provides intelligent editing based on these settings. You can also customize various aspects of the behavior of these Entry Helpers here. Such customization varies according to the file type. For example, the option to load entry helpers on opening the file and sorting attributes will not be applicable to DTD or XQuery documents.

Text View

The *Auto-complete* option automatically adds unambiguous structural components. For example, when the closing angular bracket of the start tag of an element is entered, then the end tag of that element is automatically added if this option is enabled.

In Text View, Auto-completion and entry helpers can be disabled if a file is bigger than the size specified in the *Disable Auto-completion...* combo box. This is useful if you wish to speed up the editing of large files and can do without the auto-completion feature and entry helpers. If the file size is bigger than that specified for this option, then the Text View context menu contains a toggle command for switching on and off Auto-completion and entry helper use. So you can always switch these editing aids on and off at any time during editing (in the event of files having a size greater than the size specified for this option). If the value specified for this option is smaller than the size of the opened file, locations indicated in error messages will not correctly correspond to the location in Text View.

Default copy to clipboard in grid view as

You can choose the format in which data will be exported to foreign applications using the clipboard. If you select XML-Text, the contents of the clipboard will be formatted and tagged just like the resulting XML file itself.

The structured text mode attempts to format the clipboard contents as a table, for use in a spreadsheet or database application. This option does not affect the internal clipboard format that XMLSpy uses for copying and pasting.

Table view

You can also control, how XMLSpy decides when to display repeating elements in the [Table View](#).

After making the settings, click **OK** to finish and close the Options dialog.

View

The **View** tab enables you to customize the XML documents presentation in XMLSpy.

The screenshot shows the 'View' tab of the XMLSpy Options dialog. It contains the following sections and settings:

- Enhanced Grid view**
 - ☒ Show attribute previews
 - ☐ Automatically provide optimal widths
 - ☒ Limit optimal width to pixels
 - ☐ Limit cell height to lines
- Program logo**
 - ☒ Show on start
 - ☒ Show on print
- Window title**
 - ☐ File name only
 - ☒ Full path name
- Authentic view**
 - ☒ Always open files in Authentic view when StyleVision Stylesheet assigned
- Browser view**
 - ☐ Show in a separate window by default
- Schema view**
 - ☐ Preserve content, if it still can be used in combination with the new base type
 - ☒ Confirm options on every base type modification
- Pretty-print**
 - ☒ Use Indentation

Pretty-print is used when its button is pressed in Text view or when switching or saving from all other views.

Grid View

XML elements in Grid View can be collapsed into a single line displaying the element name. When collapsed, the element's attributes can also be displayed in that line. If the *Show Attribute Previews* option is checked, attributes are displayed in gray with collapsed elements. Otherwise, attributes are not displayed with the collapsed element. Columns in the grid can be set to adjust automatically to [optimal widths](#). Additionally, the maximum optimal width and cell height can be limited. If the content of a cell is more than can fit in a cell, this is indicated by an ellipsis.

Pretty-print

When you select **Edit | Pretty-Print XML Text** in Text View or switch from another view to Text View, the XML document will be "pretty-printed". The pretty-printing will be with or without indentation according to whether the *Use Indentation* option in this dialog is checked or not. The amount of indentation can be specified in the Tabs pane of the [Text View Settings dialog](#).

Program logo

You can turn off the splash screen on program startup to speed up the application. Also, if you have a purchased license (as opposed to, say, a trial license), you will have the option of turning off the program logo, copyright notice, and registration details when printing a document from XMLSpy.

Window title

The window title for each document window can contain either the file name only or the full path name.

Authentic View

XML files based on a **StyleVision Power Stylesheet** are automatically opened in the Authentic View when this option is active.

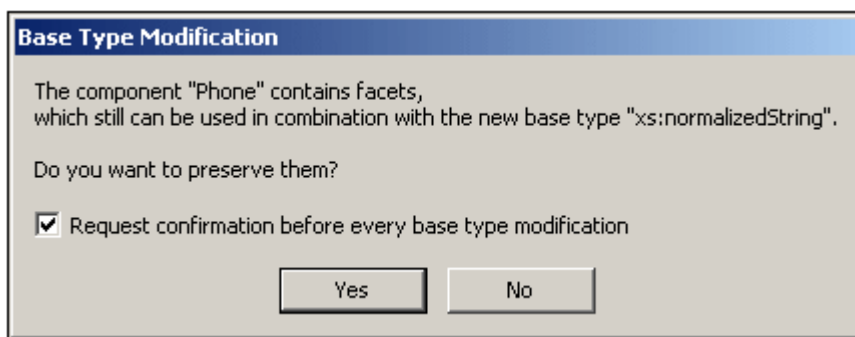
Browser View

You can choose to see the browser view in a separate window, enabling side-by-side placement of the edit and browser views.

Schema view

An XML Schema datatype can be derived from another datatype. For example, a datatype for E-mail elements can be derived from a base datatype of `xs:string` (for example, by restricting the `xs:string` datatype to a specific set of characters). If the base datatype is subsequently changed, you can set the following options:

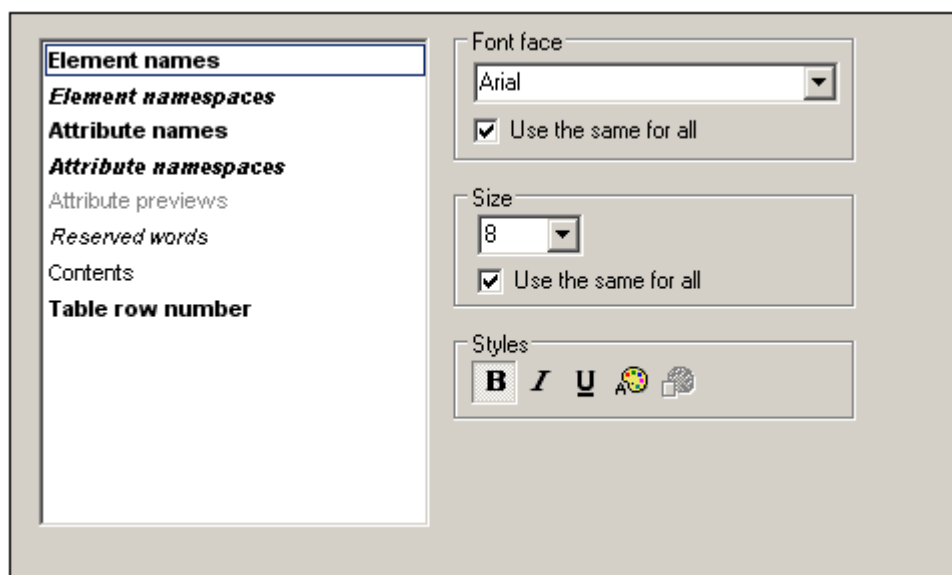
- *Preserve content*: If the definitions used to define the derived type can be used with the new base type, checking this option will automatically preserve the definitions.
- *Confirm on every modification*: After changing the base type, a dialog (see *screenshot below*) will pop up asking whether the old definitions should be preserved and used with the new base type.



After modifying the options settings, click **OK** to finish and close the Options dialog.

Grid Fonts

The **Grid fonts** tab allows you to customize the appearance of text in [Grid View](#).



Font face

You can select the font face and size to be used for displaying the various items in Grid View. The same fonts are also used for printing, so only TrueType fonts should be selected.

Size

Select the required size. If you want to use the same font size for all items, check the **Use the same for all** check box.

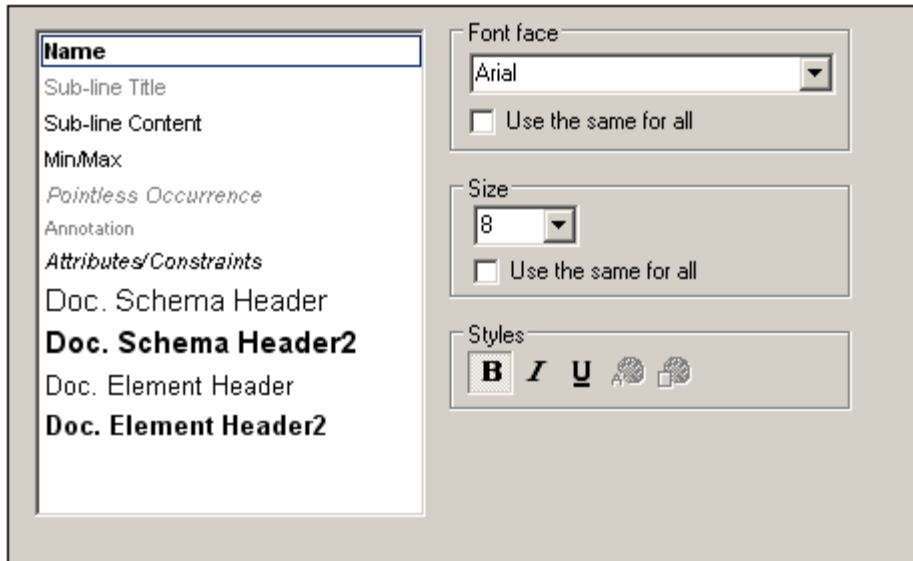
Styles

The style and color can be set using the options in this pane. The current settings are immediately reflected in the list in the left-hand pane, so you can preview the way your document will look.

After making the settings, click **OK** to finish and close the Options dialog.

Schema Fonts

The **Schema fonts** tab enables you to customize the appearance of text in [Schema View](#).



Font face

You can select the font face and size to be used for displaying the various items in the Schema Design view. The same fonts are used when printing and creating [schema documentation](#), so only TrueType fonts should be selected. Components prefixed with "Doc." are used in the schema documentation.

Size

Select the required size. If you want to use the same font size for all items, click on the Use The Same For All" check box.

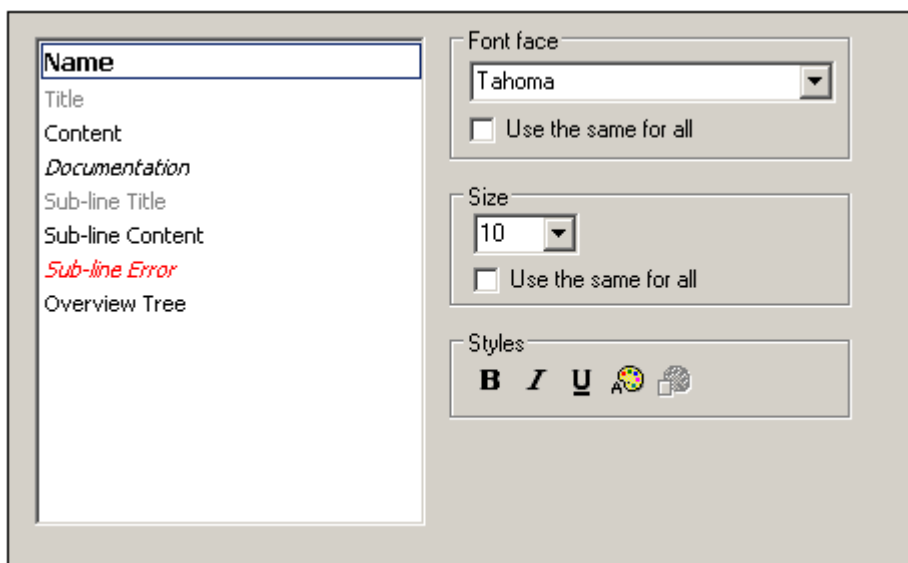
Styles

The style and color can be set using the options in this pane. The current settings are immediately reflected in the list in the left pane, so you can preview the way your document will look.

After making the settings, click **OK** to finish and close the Options dialog.

WSDL Fonts

The **WSDL fonts** tab you to customize the appearance of text in [WSDL View](#).

**Font face**

You can select the font face and size to be used for displaying the various items in the WSDL Design view. The same fonts are used when printing and creating documentation, so only TrueType fonts should be selected.

Size

Select the required size. If you want to use the same font size for all items, click on the "Use The Same For All" check box.

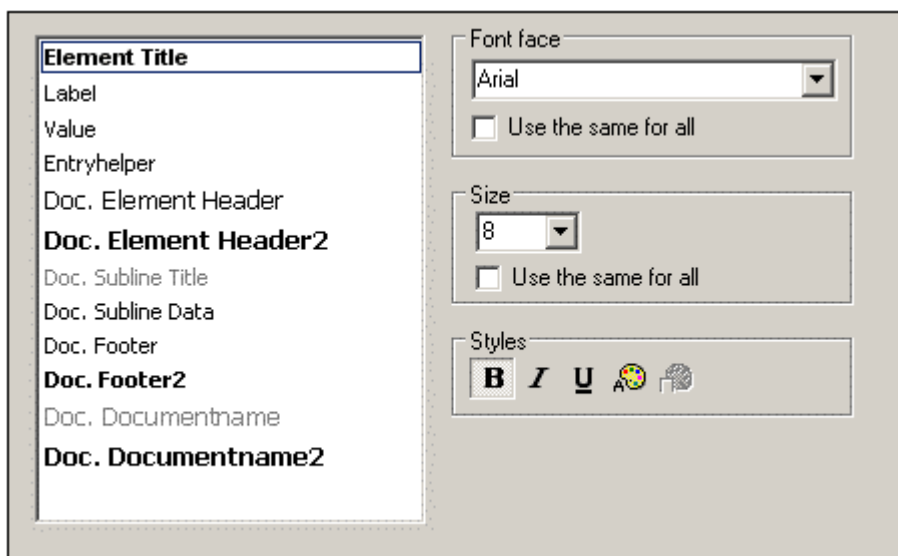
Styles

The style and color can be set using the options in this pane. The current settings are immediately reflected in the list in the left pane, so you can preview the way your document will look.

After making the settings, click **OK** to finish and close the Options dialog.

XBRL Fonts

The **XBRL fonts** tab you to customize the appearance of text in XBRL View.

**Font face**

You can select the font face and size to be used for displaying the various items in XBRL View. The same fonts are used when printing and creating documentation, so only TrueType fonts should be selected.

Size

Select the required size. If you want to use the same font size for all items, click on the *Use The Same For All* check box.

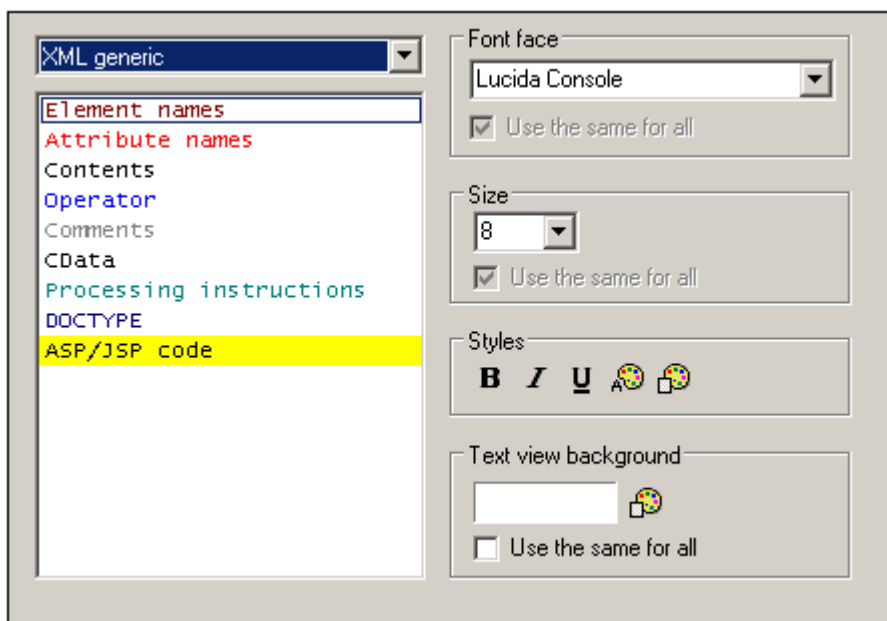
Styles

The style and color can be set using the options in this pane. The current settings are immediately reflected in the list in the left pane, so you can preview the way your document will look.

After making the settings, click **OK** to finish and close the Options dialog.

Text Fonts

The **Text fonts** tab enables you to customize the appearance of text in Text View. You can customize the appearance of text items according to the type of text item. For example, you can color element names and attribute names differently.



The text item types are categorized into three groups:

- XML generic
- XQuery
- CSS
- JSON

To customize text fonts, do the following:

1. In the combo box at top left, select the type of document for which you wish to customize text fonts. On doing this, the text item types for that document type appear in the box below the combo box. (*In the screenshot above, XML generic has been selected as the document type.*)
2. Select the text item type you wish to customize by clicking it. (*In the screenshot above, Element names has been selected.*)
3. Set the font properties using the options in the panes on the right-hand side. You can select the font-family, font-size, font-style, font-color, and background-color for the text. Additionally, you can also select a background color for the entire Text View.

Note: The same font, style, and size is used for all text item types. Only the text color and background color can be changed for individual text types. This enables the syntax coloring feature.

After making the settings, click **OK** to finish and close the Options dialog.

Colors

The **Colors** tab enables you to customize the background colors used in the Table View of Grid View. In the screenshot below, the colors have been changed from the default colors by clicking the palette icon next to each item and then selecting the preferred color.

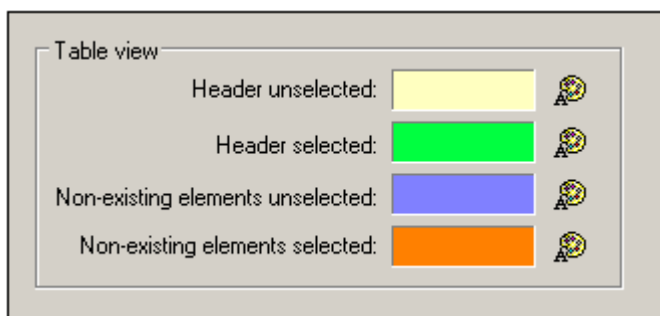


Table View

The Header unselected and Header selected options refer to the column and row headers. The screenshot below shows headers unselected; its color is as set in the dialog above.

Name Administration				
Person (3)				
	First	Last	Title	PhoneExt
1	Vernon	Callaby		582
2	Frank	Further	Accounts Receivable	471
3	Loby	Matise		963

The Header Selected color is activated when all headers are selected (*screenshot below*)—not when individual headers are selected. The screenshot below shows this using the colors defined in the dialog shown above. All headers can be selected by clicking the cell that intersects both headers or by selecting the element created as the table—or any of its ancestors.

Person (3)				
	First	Last	Title	PhoneExt
1	Vernon	Callaby		582
2	Frank	Further	Accounts Receivable	471
3	Loby	Matise		963

Non-existent Elements

When an element or attribute does not exist in the XML document, then it can be given different background colors when selected and unselected. This is shown in the screenshot below, in which the first row is selected.

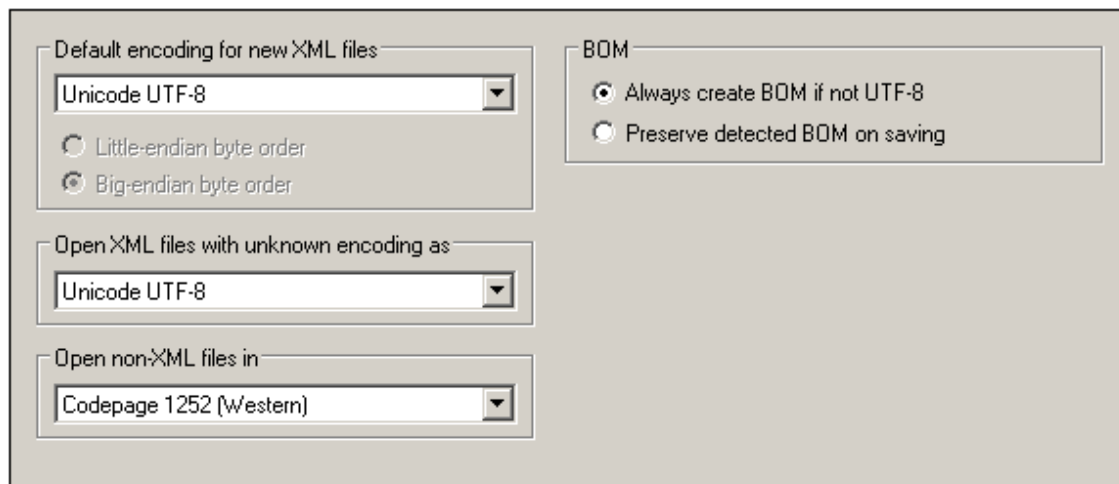
Person (3)				
	First	Last	Title	PhoneExt
1	Vernon	Callaby		582
2	Frank	Further	Accounts Receivable	471
3	Loby	Matise		963

Please note: In addition to the colors you define here, XMLSpy uses the regular selection and menu color preferences set in the Display Settings in the Control Panel of your Windows installation.

After making the settings, click **OK** to finish and close the Options dialog.

Encoding

The **Encoding** tab specifies options for file encodings.



The screenshot shows the 'Encoding' tab of a software options dialog. It contains four main sections: 1. 'Default encoding for new XML files' with a dropdown menu set to 'Unicode UTF-8' and two radio buttons for 'Little-endian byte order' (unselected) and 'Big-endian byte order' (selected). 2. 'Open XML files with unknown encoding as' with a dropdown menu set to 'Unicode UTF-8'. 3. 'Open non-XML files in' with a dropdown menu set to 'Codepage 1252 (Western)'. 4. 'BOM' with two radio buttons: 'Always create BOM if not UTF-8' (selected) and 'Preserve detected BOM on saving' (unselected).

Default encoding for new XML files

The default encoding for new XML files can be set by selecting an option from the dropdown list. A new document is created with an XML declaration containing the encoding value you specify here. If a two- or four-byte encoding is selected as the default encoding (i.e. UTF-16, UCS-2, or UCS-4) you can also choose between little-endian and big-endian byte-ordering.

The encoding of existing XML files will be retained and can only be changed with the [File | Encoding](#) command.

Open XML files with unknown encoding as

If the encoding of an XML file cannot be determined or if the XML document has no encoding specification, the file will be opened with the encoding you select in this combo box.

Open non-XML files in

Existing and new non-XML files are opened with the encoding you select in this combo box. You can change the encoding of the document by using the [File | Encoding](#) command.

BOM (Byte Order Mark)

When a document with two-byte or four-byte character encoding is saved, the document can be saved either with (i) little-endian byte-ordering and a little-endian BOM (*Always create BOM if not UTF-8*); or (ii) the detected byte-ordering and the detected BOM (*Preserve detected BOM on saving*).

After making the settings, click **OK** to finish and close the Options dialog.

XSL

The **XSL** tab (*screenshot below*) enables you to define options for [XSLT transformations](#) and [XSL-FO transformations](#) carried out from within the application.

☒ Built-in XSLT engine (Important: the built-in XSLT engine is always used for XSLT debugging)
☐ Microsoft® XML Parser (MSXML): ☐ v3.0 ☐ v4.0 ☐ v6.0 ☒ Choose version automatically
☐ External XSL transformation program:

Please enter the command line for executing an external XSL transformation program in the form:
 Program.exe %1 %2 %3
 where %1 will be replaced with the XML input file name, %2 with the output file name and %3 (optional) with XSL style-sheet file name. Feel free to add any other parameters that are required by the external program.

☒ Show external program output in Messages window after transformation
☒ Show external program error output in Messages window after transformation

Default file extension of output file: ☒ Reuse output window
☒ Use file extension from <xsl:output method=""> attribute if provided

Please enter path to XSL-FO transformation engine (if using FOP enter path to fop.bat):

☒ Use XSLT engine selected above to perform XSLT part and then XSL-FO engine for FO part
☐ Use XSL-FO engine for both XSLT and FO parts of transformation

XSLT transformations

XMLSpy contains the Altova XSLT 1.0 Engine and Altova XSLT 2.0 Engine, which you can use for XSLT transformations. The appropriate XSLT engine (1.0 or 2.0) is used (according to the value of the `version` attribute of the `xsl:stylesheet` or `xsl:transform` element). This applies both for XSLT transformations as well as for XSLT debugging using XMLSpy's XSLT/XQuery Debugger.

For transforming XML documents using XSLT, you could use one of the following:

- The built-in Altova XSLT Engine (comprising the Altova XSLT 1.0 Engine and the Altova XSLT 2.0 Engine).
- The MSXML 3.0, 4.0, or 6.0 parser (which is pre-installed). If you know which version of the MSXML parser is running on your machine, you could select it; otherwise, you should let the application select the version automatically. (The *Choose version automatically* option is active by default.) In this case, the application tries to select the most recent available version.
- An external XSLT processor of your choice. You must specify the command line string for the external XSLT processor. The following variables are available for building the command line string:

%1 = XML document to process
 %2 = Output file to generate
 %3 = XSLT stylesheet to use (if the XML document does not contain a reference to a stylesheet)

For example, the command to run a simple transformation with the Saxon (XSLT 1.0) processor is:

```
saxon.exe -o output.xml input.xml stylesheet.xslt
parameter-name=parameter-value
```

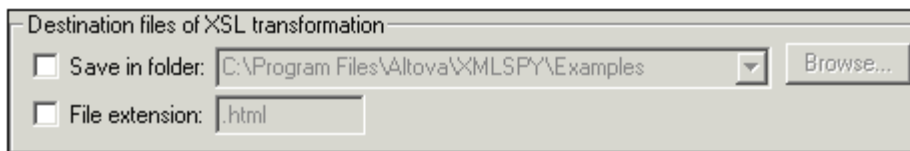
To run this command from the application, select the External XSL Transformation Program radio button, and enter the following line in the text box:

```
c:\saxon\saxon.exe -o %2 %1 %3 parameter-name=parameter-value
```

Check the respective check boxes to show the output and error messages of the external program in the Messages Window in XMLSpy.

Note: The parameters set in XMLSpy's [XSLT Input Parameters dialog](#) are passed to the internal Altova XSLT Engines only. They are not passed to any other XSLT Engine that is set up as the default XSLT processor.

The *Reuse output window* option causes subsequent transformations to display the result document in the same output window. If the XML file belongs to a project and *Reuse output window* option is disabled, the setting only takes effect if the *Save in folder* output file path (*screenshot below*) in the relevant [project properties](#) is **also** disabled.



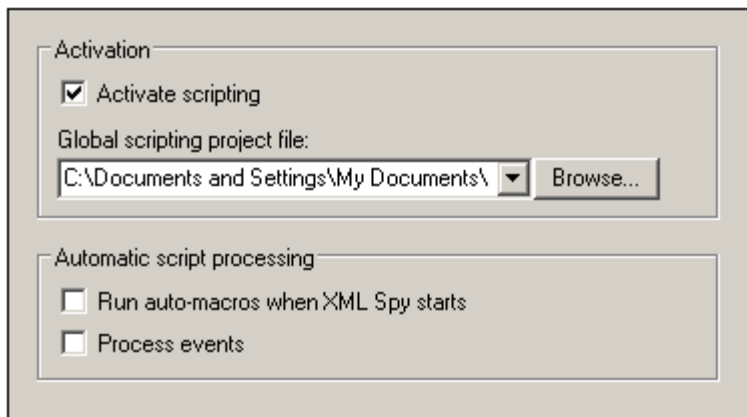
XSL-FO transformations

FO documents are processed using an FO processor, and the path to the executable of the FO processor must be specified in the text box for the XSL-FO transformation engine. The transformation is carried out using the [XSL/XQuery | XSL-FO Transformation](#) menu command. If the source file (the active document when the command is executed in the IDE) is an XSL-FO document, the FO processor is invoked for the transformation. If the source document is an XML document, an XSLT transformation is required to first convert the XML document to an XSL-FO document. This XSLT transformation can be carried out either by the XSLT engine you have specified as the default engine for the application (*see above*), or by the XSLT engine that might be built into the FO processor you have specified as the default FO processor for the application. To select between these two options, click the appropriate radio button.

After making the settings, click **OK** to finish and close the Options dialog.

Scripting

The **Scripting** tab (*screenshot below*) allows you to enable the [Scripting Environment](#) on application startup. Check the *Activate Scripting* check box to do this. You can then specify the Global Scripting Project file (*see screenshot below*).



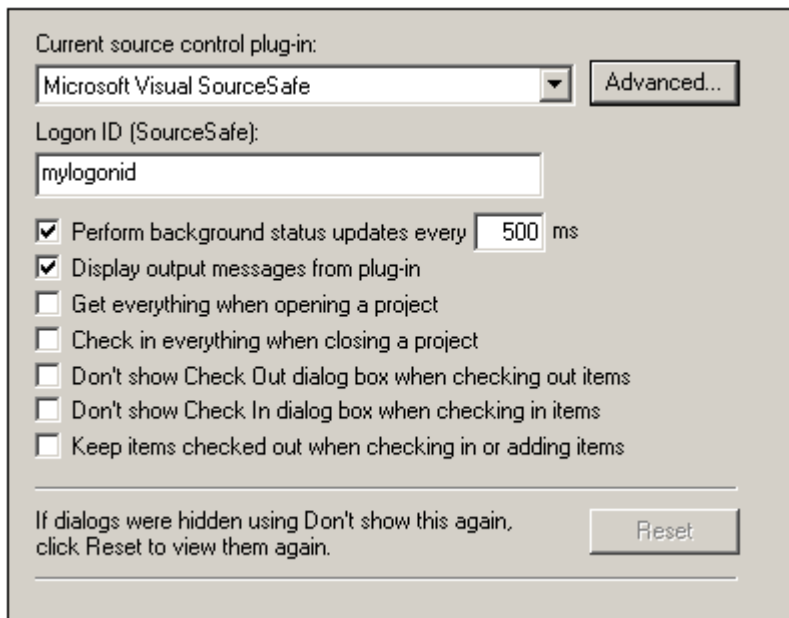
To set a global scripting project for XMLSpy, check the *Activate Scripting* check box and then browse for the Altova Scripting Project (`.asprj`) file you want. You can also specify: (i) whether

Auto-Macros in the scripting project should be automatically executed when XMLSpy starts, and (ii) whether application event handler scripts in the project should be automatically executed or not; check or uncheck the respective check boxes accordingly.

Click **OK** when done. Macros in the Global Scripting Project will then be displayed in the submenu of the **Macros** command.

Source Control

The **Source Control** tab (*screenshot below*) enables you to specify the source control provider, and the settings and default logon ID for each source control provider.



Source Control Plugin

The current source control plugin can be selected from among the currently installed source control systems. These systems are listed in the dropdown list of the combo box. After selecting the required source control, specify the login ID for it in the next text box. The **Advanced** button pops up a dialog specific to the selected source control plugin, in which you can define settings for that source control plugin. These settings are different for different source control plugins.

User preferences

A range of user preferences is available, including the following:

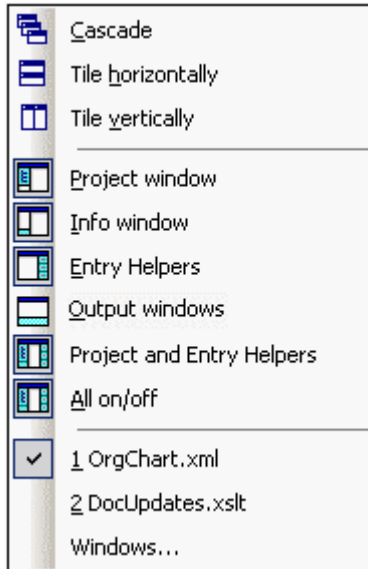
- Status updates can be performed in the background after a user-defined interval of time, or they can be switched off entirely. Very large source control databases could consume considerable CPU and network resources. The system can be speeded up, however, by disabling background status updates or increasing the interval between them..
- When opening and closing projects, files can be automatically checked out and checked in, respectively.
- The display of the Check Out and Check In dialogs can be suppressed.
- The **Reset** button is enabled if you have checked/activated the *Don't show this again* option in one of the dialog boxes. On clicking the **Reset** button, the *Don't show this*

again prompt is re-enabled.

After making the settings, click **OK** to finish and close the Options dialog.

21.17 Window Menu

To organize the individual document windows in an XMLSpy session, the **Window** menu contains standard commands common to most Windows applications.



You can cascade the open document windows, tile them, or arrange document icons once you have minimized them. You can also switch the various Entry Helper windows on or off, or switch to an open document window directly from the menu.

21.17.1 Cascade

This command rearranges all open document windows so that they are all cascaded (i.e. staggered) on top of each other.

21.17.2 Tile Horizontally

This command rearranges all open document windows as **horizontal tiles**, making them all visible at the same time.

21.17.3 Tile Vertically

This command rearranges all open document windows as **vertical tiles**, making them all visible at the same time.

21.17.4 Project Window

This command lets you switch the [Project Window](#) on or off.

This is a dockable window. Dragging on its title bar detaches it from its current position and makes it a floating window. Click right on the title bar, to allow docking or hide the window.

21.17.5 Info Window

This command lets you switch the [Info Window](#) on or off.

This is a dockable window. Dragging on its title bar detaches it from its current position and

makes it a floating window. Click right on the title bar, to allow docking or hide the window.

21.17.6 Entry Helpers

This command lets you switch all three Entry-Helper Windows on or off.

All three Entry helpers are dockable windows. Dragging on a title bar detaches it from its current position and makes it a floating window. Click right on the title bar to allow docking or hide the window.

21.17.7 Output Windows

The Output Windows are a set of tabbed output windows, such as the Messages window (which displays messages like validation results), the Find in Files window, and the XPath window (which shows XPath evaluation results). The initial setting is for them to open at below the Main Window. The Output Windows command lets you switch the Output Windows on or off.

The Output Windows window is dockable. Dragging on its title bar detaches it from its current position and makes it a floating window. Click right on the title bar to allow docking or to hide the window.

For a complete description of Output Windows see [Output Windows](#) in the section, Text View.

21.17.8 Project and Entry Helpers

This command toggles on and off the display of the Project Window and the Entry Helpers together.

21.17.9 All On/Off



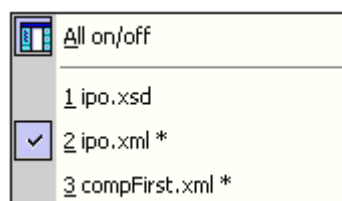
This command lets you switch all dockable windows on, or off:

- the [Project Window](#)
- the [Info Window](#)
- the three Entry-Helper Windows
- the [Output Windows](#)

This is useful if you want to hide all non-document windows quickly, to get the maximum viewing area for the document you are working on.

21.17.10 Currently Open Window List

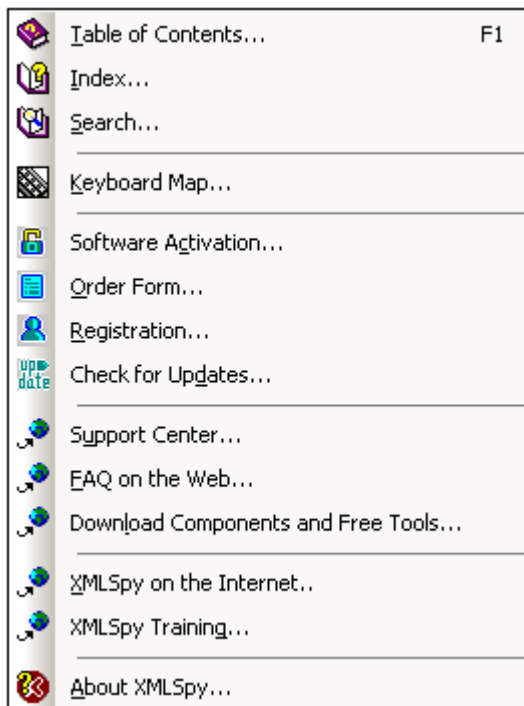
This list shows all currently open windows, and lets you quickly switch between them.



You can also use the Ctrl-TAB or CTRL F6 keyboard shortcuts to cycle through the open windows.

21.18 Help Menu

The **Help** menu contains all commands required to get help or more information on XMLSpy, as well as links to information and support pages on our web server.



The **Help** menu also contains the [Registration dialog](#), which lets you enter your license key-code once you have purchased the product.

21.18.1 Table of Contents

The **Help | Table of contents** command displays a **hierarchical representation** of all chapters and topics contained in the online help system. Use this command to jump to the table of contents directly from within XMLSpy.

Once the help window is open, use the three tabs to toggle between the table of contents, [index](#), and [search](#) panes. The Favorites tab lets you bookmark certain pages within the help system.

21.18.2 Index

The **Help | Index** command accesses the **keyword index** of the Online Help. You can also use the Index tab in the left pane of the online help system.

The index lists all relevant keywords and lets you navigate to a topic by double-clicking the respective keyword. If more than one topic matches the selected keyword, you are presented a list of available topics to choose from.

21.18.3 Search

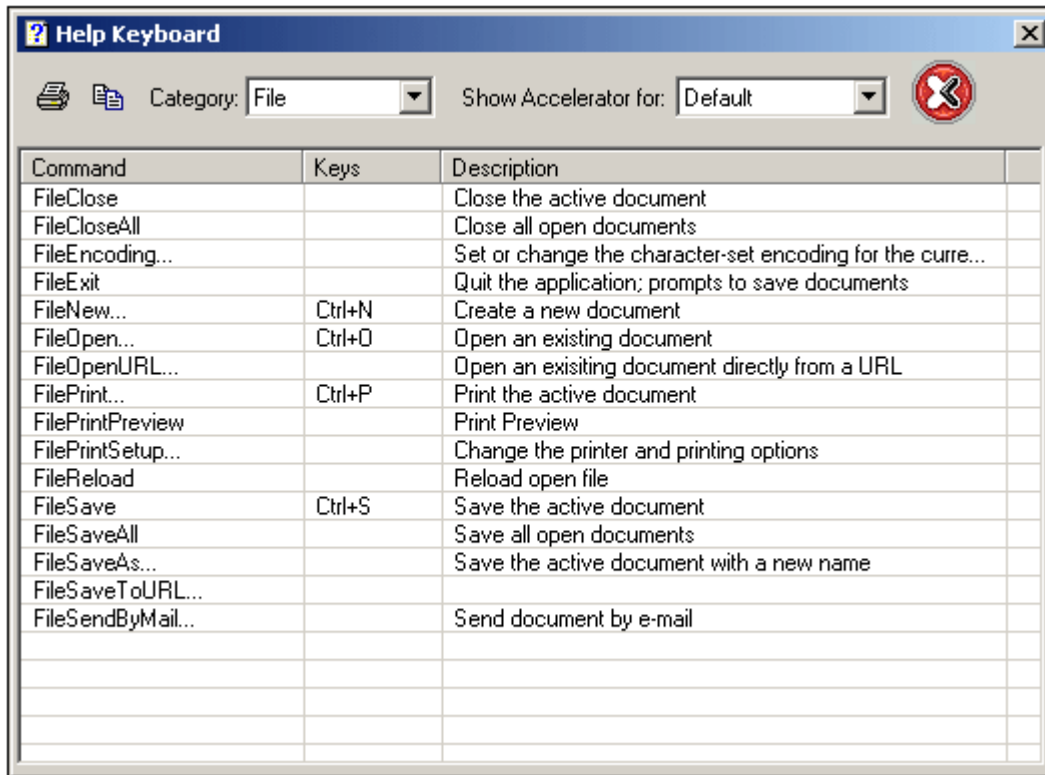
The **Help | Search** command performs a **full-text search** on the entire online help system.

1. Enter your search term in the query field and press **Enter**.
The online help system displays a list of available topics that contain the search term

- you've entered.
2. Double-click on any item in the list to display the corresponding topic.

21.18.4 Keyboard Map

The **Help | Keyboard Map...** command causes an information box to be displayed that contains a menu-by-menu listing of all commands in XMLSpy. Menu commands are listed with a description and shortcut keystrokes for the command.



To view commands in a particular menu, select the menu name in the Category combo box. You can print the command by clicking the printer icon.

You should note the following points about shortcuts:

- Certain commands (and their shortcuts) are applicable only within a certain view. For example, most of the commands in the XML menu are applicable only in Grid View. Other commands (such as **File | Save** or **XML | Check Well-Formedness**) are available in multiple views.
- Other cool shortcuts: For example, **Shift+F10** brings up the context menu in Text View and Schema View; **Ctrl+E** when the cursor is inside an element start or end tag in Text View moves the cursor to the end or start tag, respectively.
- In the [Keyboard tab](#) of the Customize dialog, you can also set your own shortcuts for various menu commands.

21.18.5 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.

21.18.6 Support Center, FAQ, Downloads

Support Center

If you have any questions regarding our product, please feel free to use this command to send a query to the Altova Support Center at any time. This is the place where you'll find links to the FAQ, support form, and e-mail addresses for contacting our support staff directly.

FAQ

To help you in getting the best support possible, we are providing a list of Frequently Asked Questions (FAQ) on the Internet, that is constantly updated as our support staff encounters new issues that are raised by our customers.

Please make sure to check the FAQ before contacting our technical support team. This will allow you to get help more quickly.

We regret that we are not able to offer technical support by phone at this time, but our support staff will typically answer your e-mail requests within one business day.

If you would like to make a feature suggestion for a future version of XMLSpy or if you wish to send us any other general feedback, please use the questionnaire form.

Download components and free tools

This command is a link to the Components Download page at the Altova website, from where you can download components, free tools, and third-party add-ins that you can use in your XML development environment. Included among these are the Altova Validator, and XSLT and XQuery engines.

21.18.7 On the Internet

On the Internet

This command takes you directly to the Altova web-server <http://www.altova.com> where you can find out about news, product updates and additional offers from the Altova team.

Training

The **Training** command takes you to the Online Training page on the Altova website. Here you can enroll for online courses to help you develop expertise in using Altova products.

21.18.8 About

This command shows the XMLSpy splash screen and copyright information dialog box, which includes the XMLSpy logo. If you are using the 64-bit version of XMLSpy, this is indicated with the suffix (x64) after the application name. There is no suffix for the 32-bit version.

Please note:

This dialog box shows the version number - to find the number of the actual build you are using, please look at the status bar, which always includes the full version and build number.

21.19 Command Line

Certain XMLSpy actions can be carried out from the command line. These commands are listed below:

Open a file

Command: xmlspy.exe file.xml

Action: Opens the file, file.xml, in XMLSpy

Note: If an XML file has an SPS file already assigned to it, then the XML file is opened in Authentic View. Otherwise, the XML file is opened in Text View. If an SPS file is not assigned, one can be assigned with the /sps flag (see below).

Open multiple files

Command: xmlspy.exe file1.xml file2.xml

Action: Opens the files, file1.xml and file2.xml, in XMLSpy

Assign an SPS file to an XML file for Authentic View editing

Command: xmlspy.exe myxml.xml /sps mysp.sps

Action: Opens the file, myxml.xml in Authentic View with mysp.sps as its SPS file. The /sps flag specifies that the SPS file that follows is to be used with the XML file that precedes the /sps flag (for Authentic View editing).

Open a new XML template file via an SPS file

Command: xmlspy.exe mysp.sps

Action: Opens a new XML file in Authentic View. The display will be based on the SPS and the new XML file will have a skeletal structure based on the SPS schema. The name of the newly created XML file must be assigned when saving the XML file.

Open an SPS file as an XML document in Text View

Command: xmlspy.exe /raw mysp.sps

Action: Opens the file mysp.sps as an XML document in Text View. The /raw flag specifies that the SPS file that follows is to be edited as an XML file.

Altova XMLSpy 2012

Programmers' Reference

Programmers' Reference

XMLSpy is an Automation Server: It exposes programmable objects to other applications called Automation Clients. An Automation Client can directly access the objects and functionality that the Automation Server makes available. So, an Automation Client of XMLSpy can use, for example, the XML validation functionality of XMLSpy. As a consequence, developers can enhance their applications with the ready-made functionality of XMLSpy.

The programmable objects of XMLSpy are made available to Automation Clients via the Application API of XMLSpy, which is a COM API. The Application API of XMLSpy will also be called Application API for short from now onwards. The object model of the Application API and a complete description of all the available objects are provided in this documentation (see the section [Application API](#)).

Execution environments

The Application API can be accessed from within the following environments:

- [Scripting Editor](#)
- [IDE Plug-ins](#)
- [External programs](#)
- [ActiveX Integration](#)

Each of these environments is described briefly below.

Scripting Editor: Customizing and modifying XMLSpy functionality

You can customize your installation of XMLSpy by modifying and adding functionality to it. You can also create Forms for user input and modify the user interface so that it contains new menu commands and toolbar shortcuts. All these features are achieved by writing scripts that interact with objects of the Application API. To aid you in carrying out these tasks efficiently, XMLSpy offers you an in-built Scripting Editor. A complete description of the functionality available in the Scripting Editor and how it is to be used is given in the [Scripting Editor](#) section of this documentation. The supported programming languages are **JScript** and **VBScript**.

IDE Plug-ins: Creating plug-ins for XMLSpy

XMLSpy enables you to create your own plug-ins and integrate them into XMLSpy. You can do this using XMLSpy's special interface for plug-ins. A description of how to create plug-ins is given in the section [XMLSpy IDE Plug-ins](#).

An application object gets passed to most methods that must be implemented by an IDE plug-in and gets called by the application. Typical languages used to implement an IDE plug-in are **C#** and **C++**. For more information, see the section [XMLSpy IDE Plugins](#).

External programs

Additionally, you can manipulate XMLSpy with external scripts. For example, you could write a script to open XMLSpy at a given time, then open an XML file in XMLSpy, validate the file, and print it out. External scripts would again make use of the Application API to carry out these

tasks. For a description of the Application API, see the section [Application API](#).

Using the Application API from outside XMLSpy requires an instance of XMLSpy to be started first. How this is done depends on the programming language used. See the section, [Programming Languages](#), for information about individual languages.

Essentially, XMLSpy will be started via its COM registration. Then the `Application` object associated with the XMLSpy instance is returned. Depending on the COM settings, an object associated with an already running XMLSpy can be returned. Any programming language that supports creation and invocation of COM objects can be used. The most common of these are listed below.

- [JScript](#) and [VBScript](#) script files have a simple syntax and are designed to access COM objects. They can be run directly from a DOS command line or with a double click on Windows Explorer. They are best used for simple automation tasks.
- [C#](#) is a full-fledged programming language that has a wide range of existing functionality. Access to COM objects can be automatically wrapped using C#.
- C++ provides direct control over COM access but requires relatively larger amounts of code than the other languages.
- [Java](#): Altova products come with native Java classes that wrap the Application API and provide a full Java look-and-feel.
- Other programming languages that make useful alternatives are: Visual Basic for Applications, Perl, and Python.

ActiveX Integration

A special case of accessing the Application API is via the XMLSpy ActiveX control. This feature is only available if the [XMLSpy integration package](#) is installed. Every ActiveX Control has a property that returns a corresponding COM object for its underlying functionality. The manager control provides an `Application` object, the document control a `Document` object, and the placeholder object, in cases where it contains the project tree, returns the `Project` object. The methods supported by these objects are exactly as described in the [Interfaces section of the Application API](#). Care must be taken not to use methods that do not make sense in the context of ActiveX control integration. For details see [ActiveX Integration](#).

About Programmers' Reference

The documentation contained in the Programmers' Reference for XMLSpy consists of the following sections:

- [Scripting Editor](#): a user reference for the Scripting Environment available in XMLSpy
- [IDE Plug-ins](#): a description of how to create plug-ins for XMLSpy
- [Application API](#): a reference for the Application API
- [ActiveX Integration](#): a guide and reference for how to integrate the XMLSpy GUI and XMLSpy functionality using an ActiveX control

1 Scripting Editor

The Scripting Editor of XMLSpy uses the Form Editor components of the Microsoft .NET Framework, and thus provides access to the Microsoft .NET Framework. This means that JScripts and VBScripts not only work with the XMLSpy API—which is a COM API and the API of XMLSpy—but can also access and use classes of the Microsoft .NET framework.

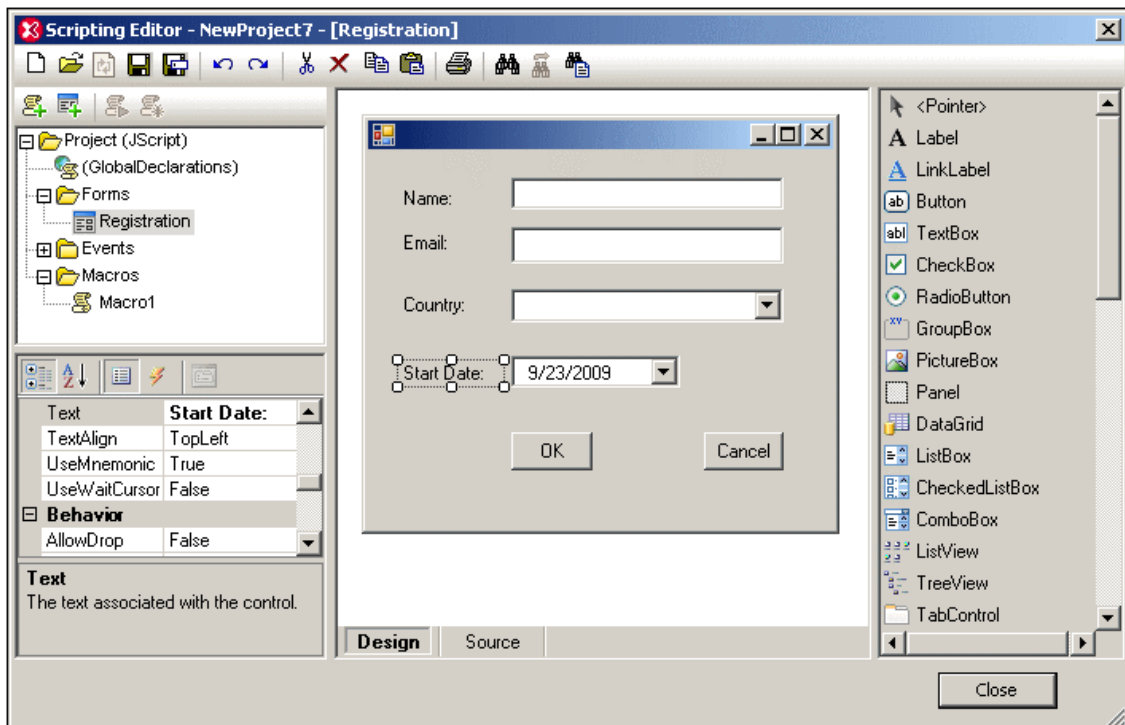
You can therefore create and use your own macros and forms within XMLSpy, and thus add to and modify the functionality of your installation of XMLSpy.

Note: Visual Basic is **not supported** as a language in the scripting environment. Only VBScript and JavaScript are. Ensure that you use VBScript syntax and not Visual Basic syntax in the scripting environment.

Note: Microsoft's **.NET Framework 2.0 or higher** is a system prerequisite for Scripting Editor, and it must be installed before XMLSpy is installed.

The Scripting Editor

The Scripting Editor (*screenshot below*) opens in a separate window and is accessed via the **Tools | Scripting Editor** menu command in the XMLSpy GUI. The programming languages that can be used in the Scripting Environment are **JScript** and **VBScript**. The scripting language can be changed by right-clicking the Project item in the Project window, selecting **Scripting Language**, and selecting the language you want.



What you can do with the Scripting Editor

In the Scripting Editor, you can create Forms, Event Handlers, and Macros to build up a

Scripting Project. A Scripting Project can then be set as the Global Scripting Project for XMLSpy, thus enabling scripts in the Scripting Project to be used in the application. Additionally, different Scripting Projects can be assigned to different XMLSpy projects, thus allowing different scripts to be used for different XMLSpy projects.

Every script project can define the .NET runtime version it wants to use. An application can handle multiple scripting projects with different .NET runtime versions simultaneously, but the appropriate .NET version must be installed. For example, script projects with .NET 4.0 will only run on computers having .NET 4.0 installed.

Documentation about the Scripting Editor

The documentation describing the Scripting Environment (this section) is organized into the following parts:

- [An overview](#), which provides a high level description of the Scripting Editor and Scripting Projects.
- [A list of steps required to create a Scripting Project](#).
- [An explanation of Global Declarations](#), together with an example.
- [A description of how to create Forms](#).
- [A discussion of XMLSpy-specific event handlers](#).
- [An explanation of how to use macros](#) in the Scripting Editor and in XMLSpy.

1.1 Overview

The Scripting Editor provides an interface in which you can: (i) graphically design Forms while assigning scripts for components in the Form; (ii) create Event Handlers, and (iii) create Macros.

These Forms, Event Handlers, and Macros are organized into scripting projects, which are then assigned to XMLSpy application projects and can be used in the application.

Variables and functions can be defined in a Global Declarations script, which is always executed before Macro or Event Handler scripts.

This section gives an overview of the Scripting Editor and Scripting Projects. It is organized into the following sections:

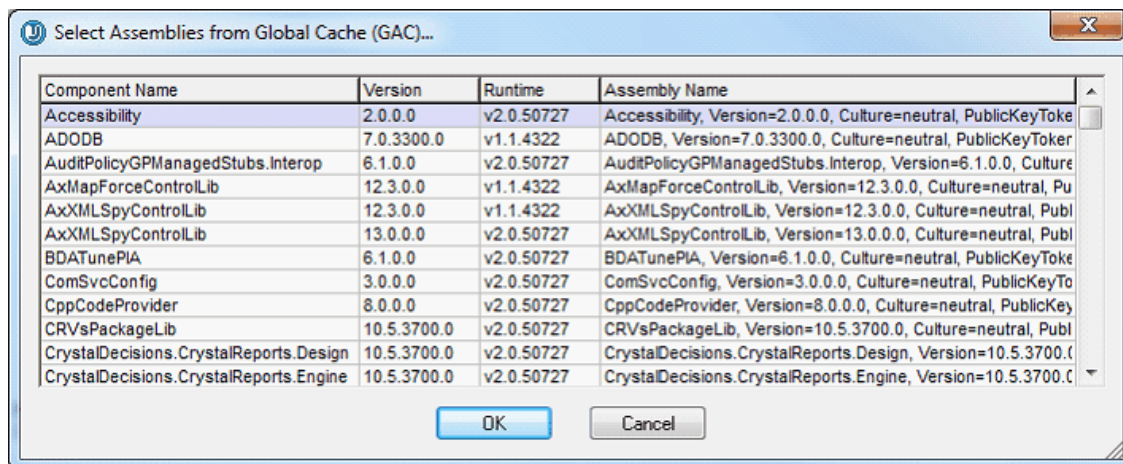
- [Scripting Projects in XMLSpy](#), which describes how the scripting projects you create with the Scripting Editor will be used in XMLSpy.
- [The Scripting Editor GUI](#), which provides a detailed look at the different parts of the Scripting Editor GUI and how they are to be used.
- [Components of a Scripting Project](#), which explains the different components that go to make up a scripting project.

The details about the creation of the various components ([Global Declarations](#), [Forms](#), [Event Handlers](#), and [Macros](#)) are described in their respective sections.

.NET assemblies

Every scripting project can have references to .NET assemblies—in addition to the default references. .NET assemblies can be added for the whole scripting project or for individual macros (by using the new `CLR.LoadAssembly` command in the source code; see [Built-in Commands](#)). Assemblies can be added, for example, from the Global Assembly Cache.

To add an assembly, right-click the project or macro, and, from the context menu that pops up, select **Add .NET Assembly | Assembly from Global Cache (GAC)**.



This works in the same way as with Visual Studio and allows access not only to the complete Microsoft .NET Framework but also to any user-defined assembly.

1.1.1 Scripting Projects in XMLSpy

All scripts and scripting information created in the Scripting Editor are stored in **Altova Scripting Projects** (.asprj files).

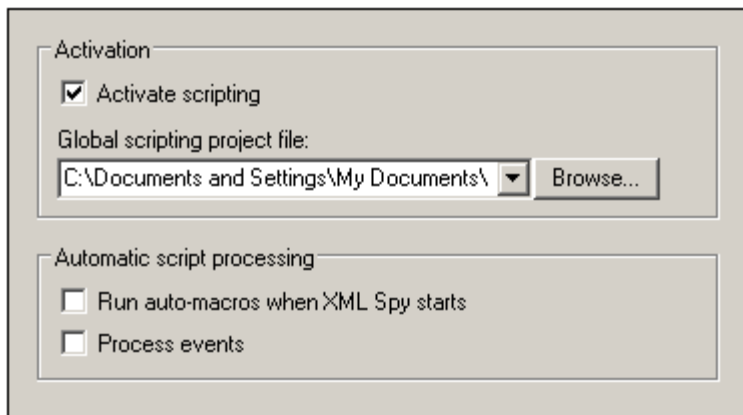
You can create any number of Altova Scripting Projects. After a scripting project has been created, it can be used in the following ways:

- It can be set as the global scripting project for XMLSpy. Scripts in the global scripting project can then be called from within the application, and macros of the Global Scripting Project can be used for all XMLSpy projects.
- It can be assigned to an XMLSpy project (as an application project). When an XMLSpy project is open in XMLSpy, scripts in the associated scripting project can be called.

Your XMLSpy package contains a sample scripting project called `SampleScripts.asprj`. This file is located in the folder: `C:\Documents and Settings\<username>\My Documents\Altova\XMLSpy2012\Examples\` and contains global declarations for a few standard tasks.

Setting the global scripting project of an application

The global scripting project of an application is set in the Scripting tab of the Options dialog of XMLSpy (*screenshot below*, **Tools | Options**).



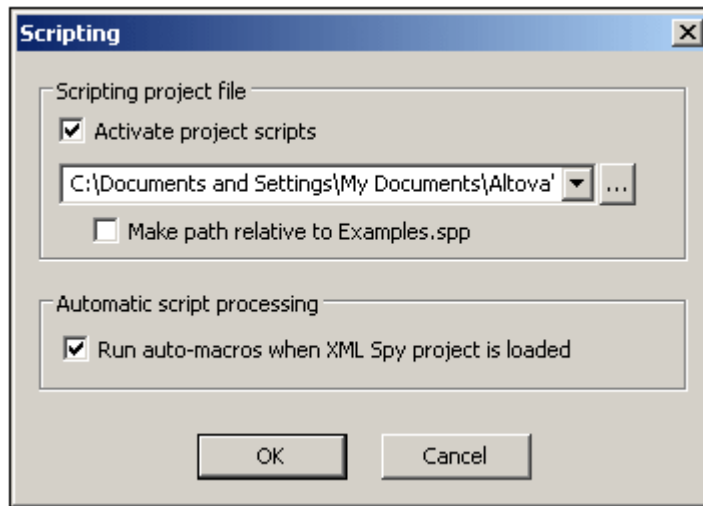
To set a global scripting project for XMLSpy, check the *Activate Scripting* check box and then browse for the Altova Scripting Project (.asprj) file you want. You can also specify: (i) whether Auto-Macros in the scripting project should be automatically executed when XMLSpy starts, and (ii) whether application event handler scripts in the project should be automatically executed or not; check or uncheck the respective check boxes accordingly.

Note: Nested script execution is possible, i.e. Macros can call other macros, and events are received during macro, or event, execution.

Assigning a scripting project to an XMLSpy project

A scripting project is assigned to an XMLSpy project as follows:

1. In the XMLSpy GUI, open the required application project.
2. Select the menu command **Project | Script Settings**. The Scripting dialog (*screenshot below*) opens.



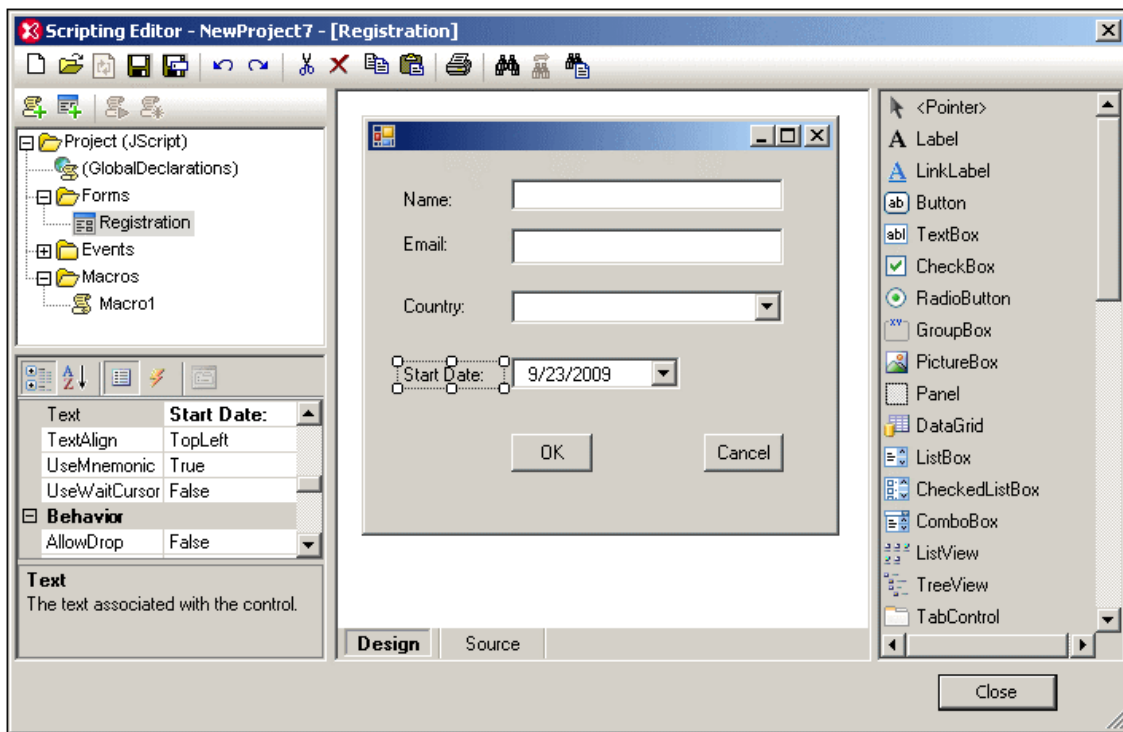
3. Check the *Activate Project Scripts* check box and select the required scripting project (.asprj file). If you wish to run Auto-Macros when the XMLSpy project is loaded, check the *Run Auto-Macros* check box.
4. Click **OK** to finish.

Note: To deactivate (that is, unassign) the scripting project of an XMLSpy project, uncheck the *Activate Project Scripts* check box.

1.1.2 The Scripting Editor GUI

The Scripting Editor GUI is shown below. It has the following parts:

- A [toolbar](#)
- A [Scripting Project Tree pane](#) (top left-hand side)
- A [Properties and Events pane](#) (bottom left)
- A [Main Window](#) with Design and Source tabs
- A [Form Object Palette](#) (right-hand side)



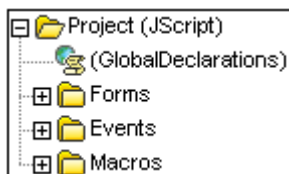
Scripting Editor toolbar

The Scripting Editor toolbar contains icons for:

- Standard file commands such as **New**, **Open**, **Save**, and **Print**. These commands are used to create new scripting projects, open existing scripting projects, and save and print scripting projects.
- Standard editing commands such as **Copy**, **Paste**, **Undo**, **Redo**, **Find**, and **Replace**. Note that the **Find** and **Replace** commands are applied to code in the Source tab of the Scripting Editor.

Scripting Project Tree

The Scripting Project Tree (*screenshot below*) shows the various components of the scripting project, structured along four main branches: (i) Global Declarations, (ii) Forms, (iii) Events, and (iv) Macros.



The Scripting Project Tree provides access to each component of the scripting project. For example, in order to display and edit a particular Form, expand the Forms folder in the tree (see *screenshot above*), right-click the Form you wish to display or edit, and click **Open** from the context menu that pops up.

A quicker way to open a Form, Event, macro, or the Global Declarations script, is to double-

click the respective icon, or text. To delete a Form or Macro from the scripting project, right-click the component and select the **Delete** command from the context menu.

The Scripting Project Tree pane contains a toolbar with icons (*screenshot below*).

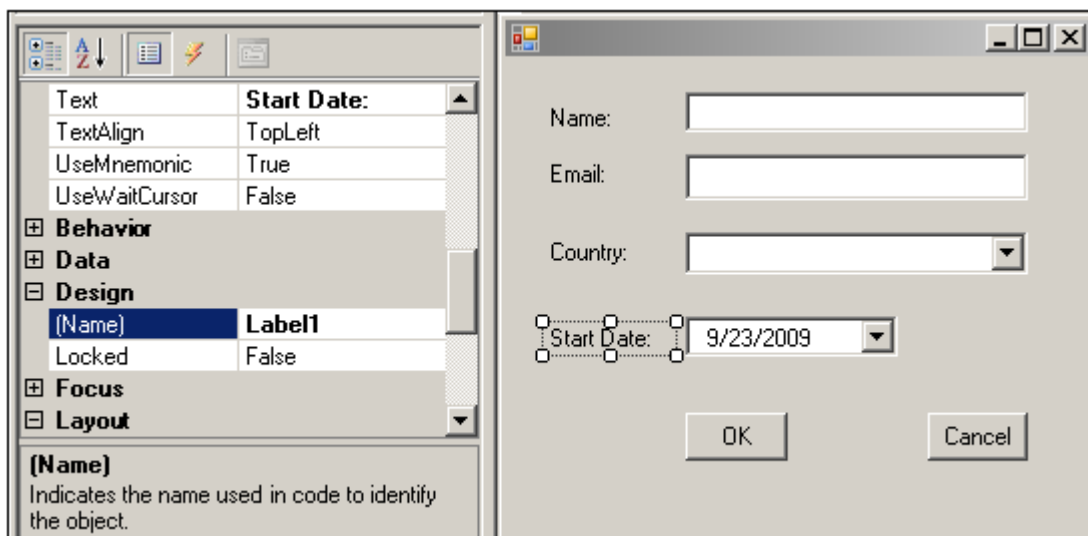


The icons, from left to right, are for: (i) [creating a new macro](#), (ii) [creating a new form](#), (iii) [running a macro](#), and (iv) [debugging a macro](#). These commands are also available in the context menu that appears when you right-click any component in the Scripting Project Tree.

Properties and Events

The Properties and Events pane (*screenshot below*) displays the following:

- Form properties, when the Form is selected
- Object properties, when an object in a Form is selected. (The screenshot below shows, at left, the properties of the object selected in the Form at right.)
- Form events, when a Form is selected
- Object events, when an object in a Form is selected



To switch between the properties and events of the selected component, click, respectively, the **Properties** icon (third from left in the Properties and Events toolbar, *see screenshot above*) and the **Events** icon (fourth from left).

The first and second icons from left in the toolbar are, respectively, the **Categorized** and **Alphabetical** icons. These display the properties or events either organized by category or organized in ascending alphabetical order.

When a property or event is selected, a short description of it is displayed at the bottom of the Properties and Events pane.

Main Window

The Main Window displays one component at a time and has one or two tabs depending on what is being displayed. If a Global Declarations script, an Event, or a Macro is being displayed,

then a single tab, the Source tab, displays the source code of the selected component.

The Source tab supports:

- syntax coloring
- source code folding
- setting/deleting bookmarks using **CTRL+F2**
- autocompletion entry helper with parameter info
- Goto Brace, Goto Brace Extend
- Zoom In / Zoom Out
- full method/property signature shown next to the autocompletion entry helper
- brace highlighting during code entry

```
if ( x == y.GetName( a, b, c() ) )
```
- mouse over popups; placing the mouse over a known method or property, displays its signature (and documentation if available)

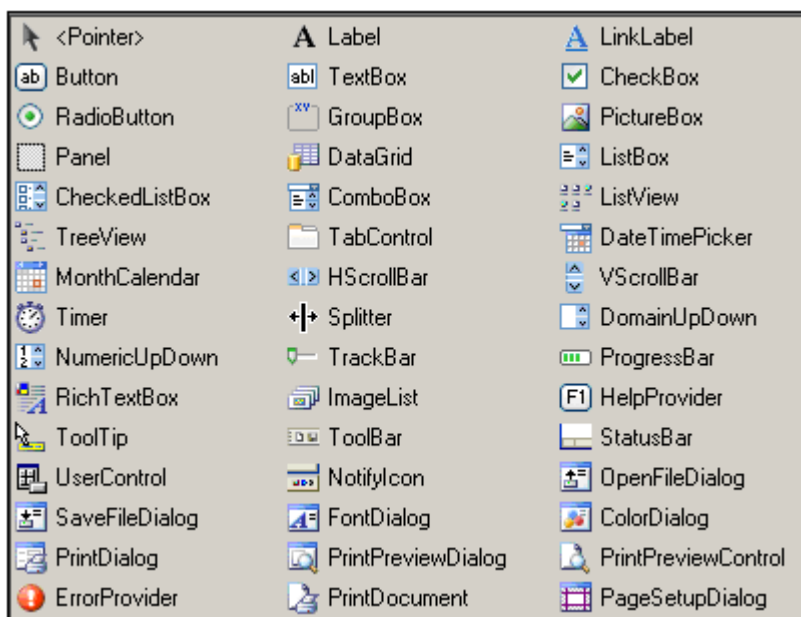
If a **Form** is being displayed, then the Main Window has two tabs: a Design tab showing and enabling the layout of the Form, and a Source tab containing the source code for the Form. Content in both the Design tab and Source tab can be edited.

Note: Since JScript and VB Script are untyped languages, entry helpers and auto-completion is supported only in cases of "fully qualified constructs" and "predefined" names.

If names start with `objDocument`, `objProject`, `objXMLData`, or `objAuthenticRange`, members of the corresponding interface will be shown. Auto-completion entry helper and parameter info are shown during editing, but can also be obtained on demand by pressing **Ctrl+Space**.

Form Object Palette

The Form Object Palette contains all the objects that are available for designing Forms and looks something like the screenshot below. Registered ActiveX controls can be added to the Form Object Palette by right-clicking the pane and selecting the **Add ActiveX Control** command



To insert an object from the Form Object Palette click the object you want in the palette, then click at the location in the Form where you wish to insert the object. The object will be placed at this location. In many cases you will need to supply some properties of the object via the Properties and Events pane. You can drag the object to other locations as well as resize it. Further, a number of editing commands, such as centering and stacking objects, can be accessed via the context menu of the selected Form object.

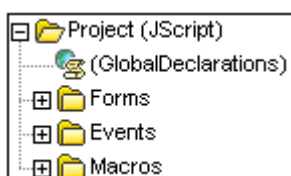
Some Form objects, such as `Timer`, are not added to the Form but are created as Tray Components in a tray at the bottom of the Main Window. You can select the object in the tray and set properties and event handlers for the object via the Properties and Events pane. For an example of how Tray Components are handled, see [Form usage and commands](#).

1.1.3 Components of a Scripting Project

An Altova Scripting Project consists of the following four major components:

- *Global Declarations*, a component which contains definitions of variables and functions that are available to, and can be used by, all Forms, Macros, and Event Handler scripts in the scripting project.
- *Forms*, a component which contains all the Forms defined in the scripting project.
- *Events*, a component which contains Event Handler scripts for all application-based—as opposed to Form-based—events.
- *Macros*, a component which contains all the Macros defined in the scripting project.

These components are displayed in and accessed via the Scripting Project Tree of the Scripting Editor (*screenshot below*).



Given below is a brief description of each of these components.

Global Declarations

The Global Declarations component is a script that contains variables and functions that can be used by Forms, Event Handlers, and Macros. The functions make use of the XMLSpy API to access XMLSpy functionality. Creating a variable or function in the Global Declarations module enables it to be accessed from all the Forms, Event Handlers and Macros in the scripting project.

To add a variable or function, open the Global Declarations component (by right-clicking it in the Scripting Project Tree and selecting **Open**) and edit the Global Declarations script in the Main Window. In this script, add the required variable or function.

Forms

In the Scripting Editor, you can build a Form graphically using a palette of Form objects such as text input fields and buttons. For example, you can create a Form to accept the input of an element name and to then remove all occurrences of that element from the active XML document.

For such a Form, a function script can be associated with a text box so as to take an input variable, and an Event Handler can be associated with a button to start execution of the delete functionality, which is available in the XMLSpy API. A Form is invoked by a call to it either within a function (in the Global Declarations script) or directly in a Macro. For details of how to create and edit Forms, see the [Forms](#) section.

Event handling

Event Handler scripts can be associated with a variety of available events. You can control events that occur both within Forms ([Form events](#)) and within the general application interface ([application events](#)). The script associated with an event is executed immediately upon the triggering of that event.

Most events have parameters which provide detailed information about the event. The return value from the script typically instructs the application about how to continue its processing (for example, the application may not allow editing).

An Event Handler runs when the relevant event occurs in the Form or in XMLSpy. For details about how to create event handlers, see [Event Handlers](#).

Macros

Macros are used to implement complex or repetitive tasks. Macros do not use either parameters or return values.

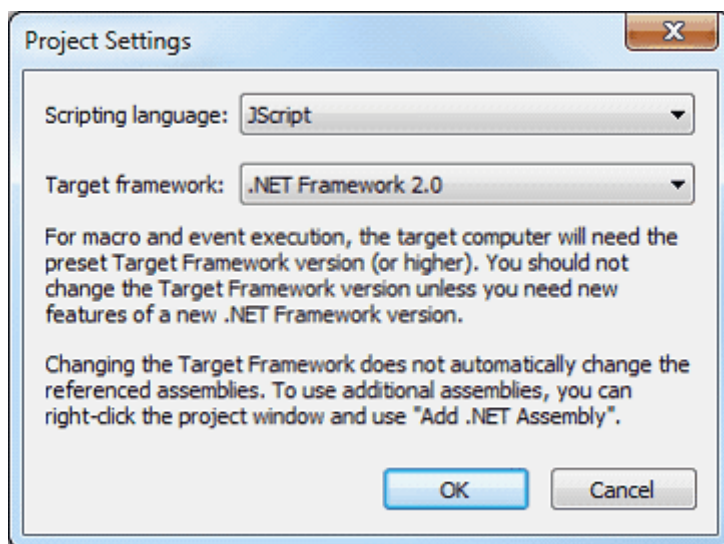
In a Macro, it is possible to access all variables and functions declared in the Global Declarations and to display Forms for user input.

For a simple example of creating a Macro, see [Writing a Macro](#). Also see [Running Macros](#) for a description of the ways in which a Macro can be called. A Macro is run from within the XMLSpy interface by clicking **Tools | Macros | [MacroName]**

1.2 Creating a Scripting Project

The broad steps for creating a Scripting Project are as follows:

1. Open the Scripting Editor by clicking the command **Tools | Scripting Editor**.
2. In the Scripting Editor, open a new scripting project by clicking the **New** icon in the Scripting Editor toolbar. The Project Settings dialog (*screenshot below*) pops up.



Select either JScript or VBScript in the first combo box and the .NET Framework in the second combo box, and click **OK**. The new Scripting Project is created.

3. Click the **Save** icon in the Scripting Editor toolbar to save the Scripting Project as a .asprj file.
4. A Scripting Project can be considered to be made up of several components that work together. These components will typically be a combination of: Global Declarations, Forms, Events, and Macros. They can be created in any order, but you should clearly understand how they work together. The way each type of component is called and executed is [described below](#). How to create each type of component is described in the respective sections about the component type.
5. After you have finished creating all the required components, save the Scripting Project (by clicking the **Save** icon in the Scripting Editor toolbar).
6. Close the Scripting Editor.

Please note:

Right clicking the Project folder and selecting "Scripting Language..." lets you change the scripting language at any time.

How Forms, Event Handlers, and Macros are called and executed

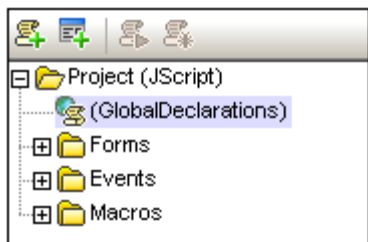
Forms, Event Handlers, and Macros are all created in the Scripting Editor. However, the way they are called and executed is different for each and has a bearing on how you create your scripting projects.

- A Form is invoked by a call to it either within a function in the Global Declarations script or directly in a Macro.
- An Event Handler runs when the relevant event occurs in XMLSpy. If an Event Handler for a single event is defined in both the Global Scripting Project and the XMLSpy -project-specific Scripting Project, then the event handler for the project-specific Scripting Project is executed first and that for the Global Scripting Project immediately afterwards.

- A Macro is executed from within the XMLSpy interface by clicking **Tools | Macros | *[MacroName]***. In a Macro, it is possible to access all variables and functions declared in the Global Declarations and to display Forms for user input.

1.3 Global Declarations

The Global Declarations component is present by default in every Scripting Project (see *screenshot below*), and therefore does not have to be created. In order to add variables and functions to the Global Declarations script of a Scripting Project, you need to open the Global Declarations script and add the code fragment to the Global Declarations script. See [Components of a Scripting Project](#) and [Creating a Scripting Project](#) for more information.



To open the Global Declarations script of a Scripting Project, right-click the *Global Declarations* item in the Scripting Project Tree (*screenshot above*), and select **Open**. The Global Declarations script opens in the Main Window.

Note: Every time a macro is executed or an event handler is called, global declarations are re-initialized.

Given below is an example function. Remember that creating a variable or function in the Global Declarations script makes this variable or function accessible to all Forms, Event Handlers, and Macros.

Example function

A function called `RemoveAllNamespaces` would have code like this:

```
function RemoveAllNamespaces( objXMLData)
{
    if( objXMLData == null)
        return;

    if( objXMLData.HasChildren)    {
        var objChild;

        // spyXMLDataElement := 4
        objChild = objXMLData.GetFirstChild(4);

        while( objChild) {
            RemoveAllNamespaces( objChild);

            try{
                var nPos, txtName;
                txtName = objChild.Name;

                if( ( nPos = txtName.indexOf(":") ) >= 0)  {
                    objChild.Name = txtName.substring( nPos+1);
                }

                objChild = objXMLData.GetNextChild();
            }
            catch( Err)    {
                objChild = null;
            }
        }
    }
}
```

```
    }  
}
```

Note:

- It is possible to define local variables and helper functions within macros and event handlers. Example:

```
//return value: true allows editing  
//return value: false disallows editing  
var txtLocal;  
function Helper()  
{  
    txtMessage = txtLocal;  
    Application.ShowForm( "MsgBox" );  
}  
function On_BeforeStartEditing( objXMLData )  
{  
    txtLocal = "On_BeforeStartEditing()";  
    Helper();  
}
```

- Recursive functions are supported.

1.4 Forms

Creating and editing Forms in the Scripting Editor consists of the following steps:

1. [Creating a New Form](#). The new Form is created and named, and has properties defined for it.
2. [Designing the Form](#). A Form is designed by adding Form Objects to it and assigning values for the different Form Objects.
3. [Scripting Form Events](#). Scripts are assigned to Form-related events.

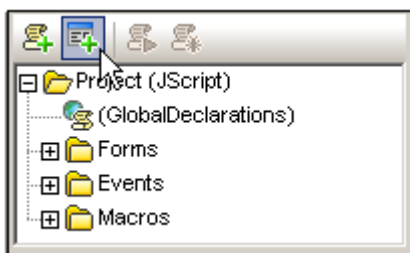
1.4.1 Creating a New Form

Creating a new Form in the Scripting Editor involves the following steps:

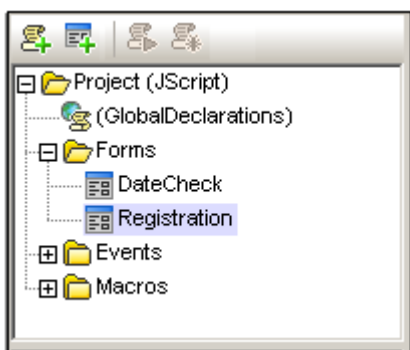
1. [Creating a new Form and naming it](#)
2. [Specifying the properties of the Form](#)

Creating a new Form and naming it

To add a new Form to a scripting project, click the **Add Form** icon (*highlighted in screenshot below*) in the toolbar of the Project Overview pane. Enter the name of the new Form.

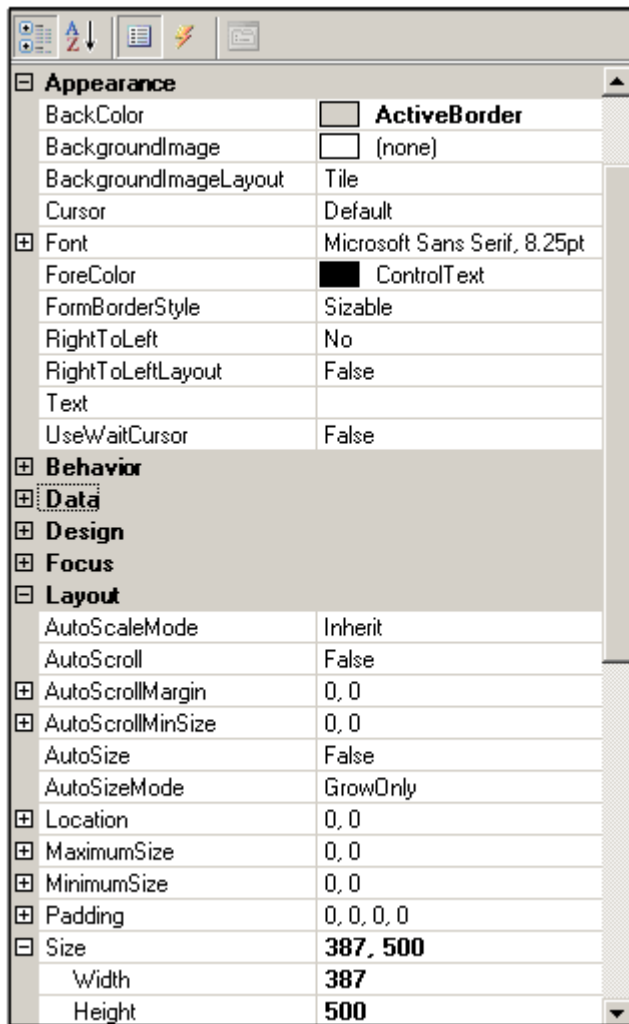


A new Form is added to the project. It appears in the Main Window and an entry for it is created in the Scripting Project Tree pane, under the Forms heading. Press the F2 function key to rename the form, or right click the form name and select Rename from the context menu. In the screenshot below, we have named the new Form *Registration*.



Form properties

The properties of the Form, such as its size, background color, and font properties, can be set in the Properties pane. The screenshot below shows the size and background-color property values in bold, in the *Layout* and *Appearance* categories, respectively.



Testing a Form

You can test a form in the Scripting Editor by right-clicking it in the Project Overview pane and selecting the **Test Form** Command.

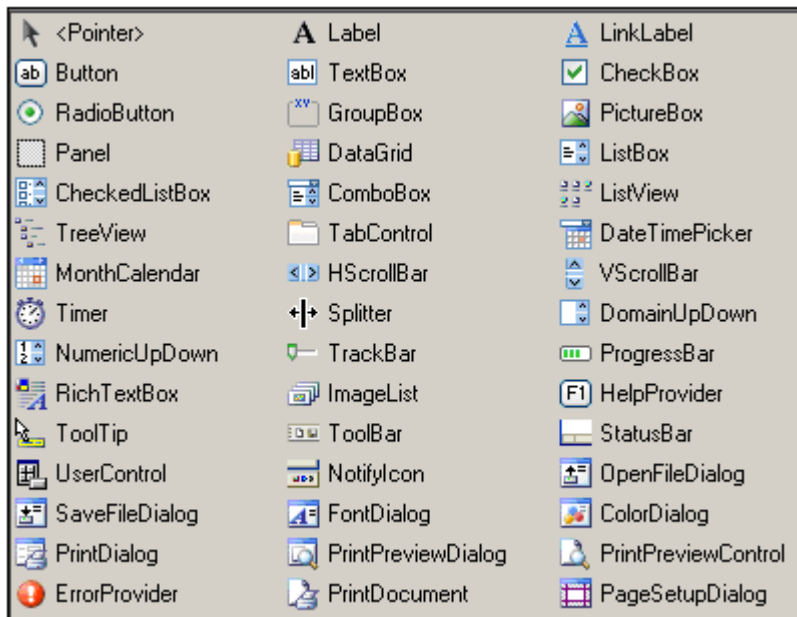
1.4.2 Form Design and Form Objects

Designing a Form consists of the following steps:

- Placing an object from the [Form Object Palette](#) in the Form design.
- Assigning values for the [properties of individual Form Objects](#).
- [Assigning scripts for Form-based events](#).

The Form Object Palette

The Form Object Palette contains all the objects that are available for designing Forms and looks something like the screenshot below. Registered ActiveX controls can be added to the Form Object Palette by right-clicking the pane and selecting the **Add ActiveX Control** command



To insert an object from the Form Object Palette click the object you want in the palette, then click at the location in the Form where you wish to insert the object. The object will be placed at this location. In many cases you will need to supply some properties of the object via the Properties and Events pane. You can drag the object to other locations as well as resize it. Further, a number of editing commands, such as centering and stacking objects, can be accessed via the context menu of the selected Form object.

Some Form objects, such as `Timer`, are not added to the Form but are created as Tray Components in a tray at the bottom of the Main Window. You can select the object in the tray and set properties and event handlers for the object via the Properties and Events pane. For an example of how Tray Components are handled, see [Form usage and commands](#).

Some of the most commonly used objects are described below:



Label: Adds text fields such as captions or field descriptions.



Button: Adds a button. It is possible to assign bitmaps as background images for these buttons.



Check Box: Adds a check box, which enables *Yes/No* type selections.



Combo Box: Adds a combo box, which allows the user to select an option from a drop-down menu.



List Box: Adds a list box, which displays a list of items for selection.



TextBox: Enables the user to enter a single line of text.



Rich TextBox: Enables the user to enter multiple lines of text.

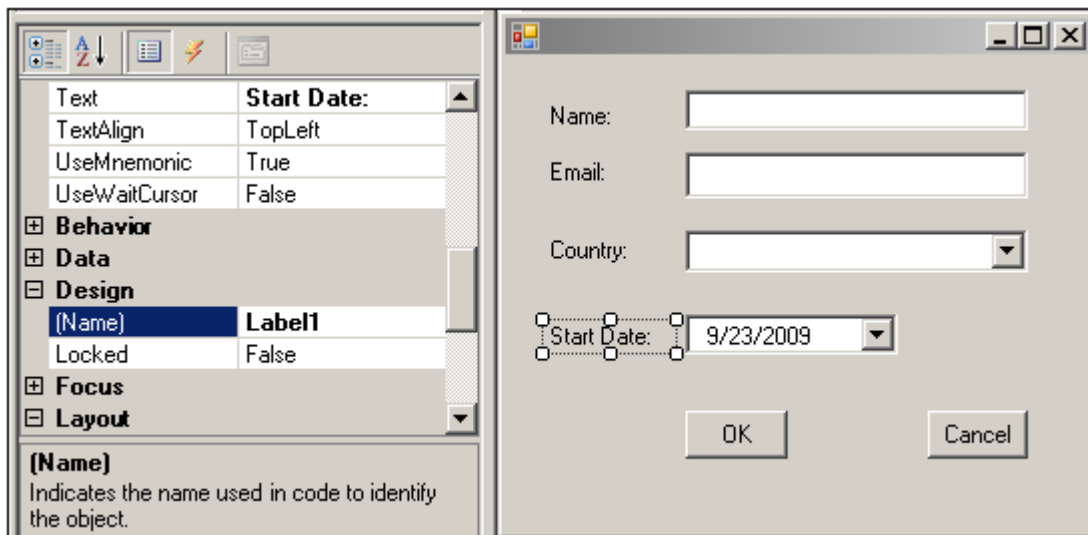
Creating objects and setting their properties

To create an object in the Form, first select the required object in the Form Object Palette and then click the location in the Form where you want to insert it. After the object has been inserted, you can resize it as well as drag it to another location in the Form.

When an object is selected in the design, you can specify its properties in the Properties and Events pane. In the toolbar of the Properties and Events pane, click the Properties icon to display a list of the object's properties.

For example, in the screenshot below, the Label object with the text *Start Date* has been selected in the design. In the Properties and Events pane, the name of the object (which is the name that is to be used to identify the object in code, `Label1` in the screenshot below) is given in the *Design* category of properties; in this case, the name of the object is `Label1`.

The text of the label (which is what appears in the Form) must be entered as the value of the *Text* property in the *Appearance* category of properties.



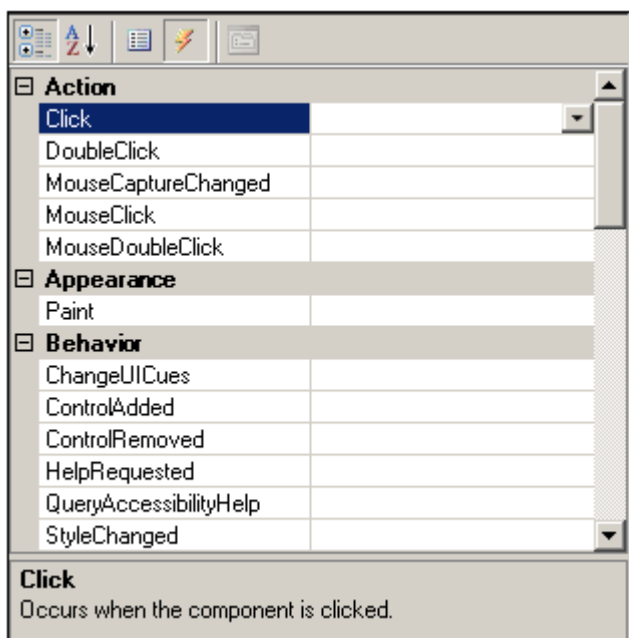
To assign other object properties, enter values for them in the Properties and Events pane.

Testing a Form

You can test a form in the Scripting Editor by right-clicking it in the Project Overview pane and selecting the **Test Form** Command.

1.4.3 Form Events

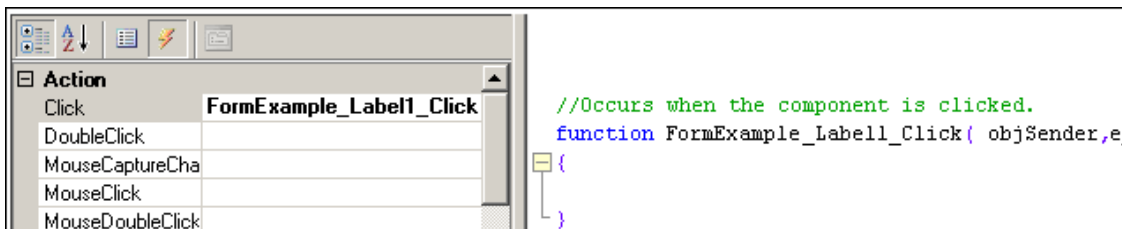
When an object is selected in the design, clicking on the Events icon in the toolbar of the Properties and Events pane (*fourth icon from left*), displays all the events available for that object (see *screenshot below*). These can be displayed either by category (*screenshot below*) or alphabetically.



For each event, you can enter the name of an existing event handler or function. Alternatively:

- you can double click on an event to create: (i) an empty function script in the *Source* tab of the Main Window, and (ii) an association of the newly created function with the selected event.
- double click a button in the design tab, to directly generate the handler stub in the code window.

The screenshot below was taken after the *Click* event was double-clicked. Notice that an empty event handler function called `FormExample_Label1_Click` has been created in the Main Window and that, in the Properties and Events pane, this function has been associated with the *Click* event.



Enter the required scripting code and save the project.

Writing the required scripts

After the visual design of the form is complete, form objects will typically be associated with suitable scripts. The example below is a script that adds colors when a button is clicked. The script is inserted as an event handler for the *Click* event of the button `Button1` (the event is available in the Properties and Events pane when the button is selected in the design):

```
function FormExample_Button1_Click( objSender, e_EventArgs )
{
    // Sets the ForeColor (red) of the button.
    objSender.ForeColor = CLR.Static( "System.Drawing.Color" ).Red;
    // Sets the BackColor (blue) of the button.
```

```
objSender.BackColor = CLR.Static( "System.Drawing.Color" ).Blue;  
// Sets the form BackColor (green).  
objSender.FindForm().BackColor = CLR.Static( "System.Drawing.Color"  
).Green;  
}
```

1.5 Events

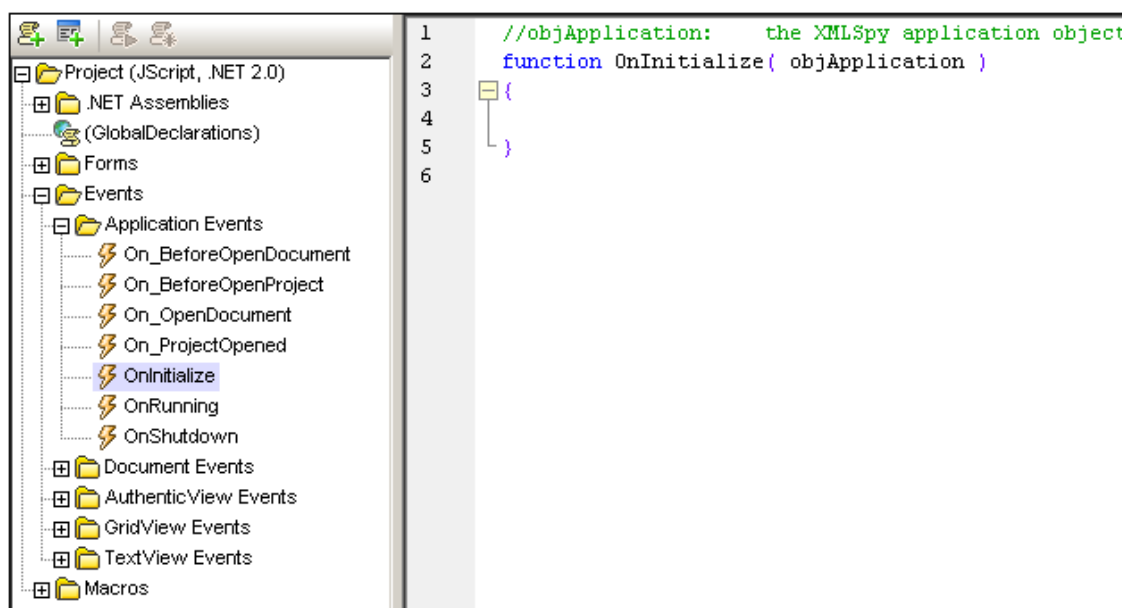
The Events folder of the scripting project (*see screenshot below*) contains folders for the following type of events:

- Application Events
- Document Events
- Authentic View Events
- Grid View Events
- Text View Events

Note that these events are XMLSpy-specific, as opposed to Form-based events. Each of the folders listed above contains a set of events for which Event Handler scripts can be written.

Application Events, for example, are shown in the screenshot below.

To access the event handler script of any of these events, right-click the event and select **Open** from the context menu. The script will be displayed in the Main Window (*see screenshot below*) and can be edited there. After you have finished editing the script, save changes by clicking the **Save** command in the toolbar of the Scripting Editor.



Note the following points:

- Event Handlers need function headers with the correct spelling of the event name. Otherwise the Event Handler will not be called.
- It is possible to define local variables and helper functions within Macros and Event Handlers. Example:

```
//return value: true allows editing
//return value: false disallows editing
var txtLocal;
function Helper()
{
    txtMessage = txtLocal;
    Application.ShowForm( "MsgBox" );
}
function On_BeforeStartEditing( objXMLData )
{
```

```

        txtLocal = "On_BeforeStartEditing()";
        Helper();
    }

```

- In order for events to be processed, the Process Events options must be toggled on in the Scriptings options of XMLSpy. See [Scripting Projects in XMLSpy](#) for details.
- Also see [Programming Points](#).

Application Events

OnInitialize

The `OnInitialize` event is raised after the main window becomes visible but before any project is loaded. This event is not raised if the application can't be loaded at all.

OnRunning

If the application is completely loaded and after the `OnInitialize` event occurs, the `OnRunning` event is raised.

OnShutdown

The event is raised after any open project and all documents have been closed on shutdown of the application. The main window is no longer visible.

Example

The following script is an Event Handler for the `On_BeforeOpenProject` event. It allows you to add a script that will be executed each time before XMLSpy opens a project. The example script below sequentially opens all XML files located in the XML folder of the project and validates them. If the validation fails, the script shows the validation error and stops. If a file passes the validity test, it will be closed and the next file will be opened.

Enter the following script for the `On_BeforeOpenProject()` event, and then save the scripting project.

```

function On_BeforeOpenProject()
{
    var bOK;
    var nIndex, nCount;
    var objItems, objXMLFolder = null;

    objItems = Application.CurrentProject.RootItems;
    nCount = objItems.Count;

    // search for XML folder
    for( nIndex = 1; nIndex <= nCount; nIndex++) {
        var txtExtensions;
        txtExtensions = objItems.Item( nIndex ).FileExtensions;

        if( txtExtensions.indexOf( "xml" ) >= 0 )    {
            objXMLFolder = objItems.Item( nIndex );
            break;
        }
    }

    // does XML folder exist?
    if( objXMLFolder ) {
        var objChild, objDoc;

        nCount = objXMLFolder.ChildItems.Count;

        // step through associated xml files
        for( nIndex = 1; nIndex <= nCount; nIndex++) {

```

```

objChild = objXMLFolder.ChildItems.Item( nIndex );

try{
    objDoc = objChild.Open();

    // use JScript method to access out-parameters
    var strError = new Array(1);
    var nErrorPos = new Array(1);
    var objBadData = new Array(1);

    bOK = objDoc.IsValid( strError, nErrorPos, objBadData );

    if(! bOK) {
        // if the validation fails, we should display the
        // message from XMLSpy
        // of course we have to create the form "MsgBox" and
        // define the global txtMessage variable
        //
        // txtMessage = Position:" + nErrorPos[0] + "\n" + // strError[0];
        // txtMessage += "\n\nXML:\n" + objBadData[0].Name + ", " +
        //     objBadData[0].TextValue;
        //
        // Application.ShowForm( "MsgBox" );

        break;
    }

    objDoc.Close( true );
    objDoc = null;
}
catch( Err ) {
    // displaying the error description here is a good idea

    // txtMessage = Err.Description;
    // Application.ShowForm( "MsgBox" );

    break;
}
}
}

```

Testing the Event Handler

Switch to XMLSpy, and open a project to see how the `BeforeOpenProject` event is handled.

1.6 Macros

Macros automate repetitive or complex tasks. In the Scripting Environment, you can create a script that calls application functions as well as custom functions that you have defined. This flexibility provides you with a powerful method of automating tasks within XMLSpy. This section about macros is organized as follows:

- [Creating and Editing a Macro](#) describes how to create a new macro and edit an existing one.
- [Running a Macro](#) explains how a macro can be run from the Scripting Editor and from the broader XMLSpy environment as well.
- [Debugging](#) describes how macros can be debugged.

Key points about macros

Given below is a summary of important points about macros.

- Any number of macros can be added to the active scripting project. These macros are saved in the Altova Scripting Project file (.asprj file).
- Functions that are used in a macro can be saved as a Global Declaration. All Global Declarations are also saved in the Altova scripting project file (.asprj file).
- The macro can be tested by running it from within the Scripting Editor, and it can be debugged from within the Scripting Editor.
- XMLSpy can have one global Scripting Project, and a second scripting project, assigned to the currently loaded project, active at any one time; the macros are available to both of them. See [Running a Macro](#) for details.

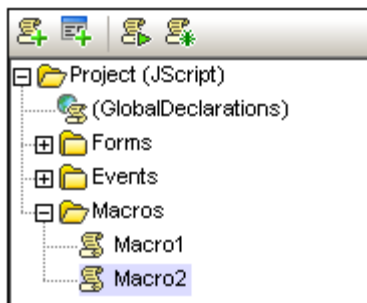
1.6.1 Creating and Editing a Macro

The following operations enable you to create a new macro and edit an existing macro.

Creating a new macro

Right-click the Macro folder in the Scripting Projects tree and select **Add Macro** from the context menu. (The **Add Macro** command can also be selected from the context menu of any item in the Scripting Projects tree.) Alternatively, click the **New Macro** icon in the toolbar of the Scripting Projects tree.

The newly created (and empty) macro document is displayed in the Main Window, and the name of the macro is displayed in the title bar of the Scripting Editor (*screenshot below*).



Naming or renaming a macro

To name or rename a macro, click the macro name in the Scripting Project tree and press the **F2** function key, or right click the name and select **Rename** from the context menu.

Opening a macro

To open a macro, right-click the macro in the Macros folder of the Scripting Project tree (see *screenshot above*), and select the **Open** command. The macro is displayed in the Main Window and its name is displayed in the title bar of the Scripting Editor (*screenshot below*). Alternatively, double-clicking a macro in the Scripting Project tree opens it in the Main Window.



Editing the macro

To edit a macro, enter or edit its code in the Main Window. For example, the following code creates the Form named `Form1` in memory and then shows it. `Form1` must already have been created (using the Scripting Editor's [Form creation](#)) before this macro is run.

```
objForm = CreateForm( 'Form1' );  
objForm.ShowDialog();
```

The following macro uses the `RemoveAllNamespaces` function to remove all namespaces in the active XML document.

```
if( Application.ActiveDocument != null) {  
  
    RemoveAllNamespaces( Application.ActiveDocument.RootElement);  
    Application.ActiveDocument.UpdateViews();  
}
```

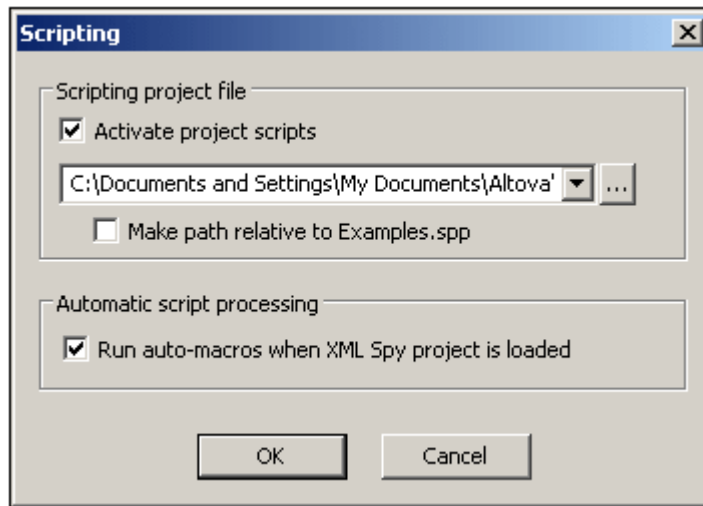
The `RemoveAllNamespaces` function itself will have to be defined in the Global Declarations script. After the `RemoveAllNamespaces` function has been defined, the macro is complete and can be run.

Note: Macros do not support parameters or return values.

Setting a macro as an Auto-Macro

When a macro is set as an Auto-Macro it can be run automatically when: (i) XMLSpy is started, or (ii) an Altova XMLSpy project is loaded in XMLSpy. To specify whether Auto-Macros should be run in each of these two events, check the *Run Auto-Macros* option in the Automatic Script Processing pane of the relevant dialogs:

- *When XMLSpy is started:* the Scripting tab of the XMLSpy Options dialog (**Tools | Options** menu command).
- *When an XMLSpy project is loaded into XMLSpy:* the Scripting dialog (*screenshot below*, **Project | Scripting Settings** menu command).



To set a macro as an Auto-Macro, right-click the macro in the Scripting Project tree and select the command **Set as Auto-Macro**. This is a toggle command; so to remove the Auto-Macro setting of a macro, select the command again.

1.6.2 Running a Macro

To run a macro in the Scripting Editor, right-click the macro in the Scripting Project tree and select the command **Run Macro**.

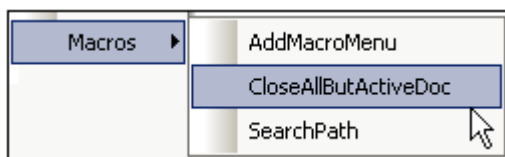
There are different ways to run a macro from XMLSpy:

- [Via the Tools | Macros menu](#) of XMLSpy.
- [By creating and using a toolbar button](#) for a macro.
- [By creating and using a menu item](#) for a macro.

Note that only one macro can be run at a time. After a macro (or event) is executed, the script is closed and global variables lose their values.

The XMLSpy command to run Macros

The **Tools | Macros** menu command (*screenshot below*) opens a submenu containing the macros defined in the Scripting Project that is currently active in XMLSpy. The active Scripting Projects are specified in the Scripting tab of the Options dialog, or in the Scripting tab of the project settings.

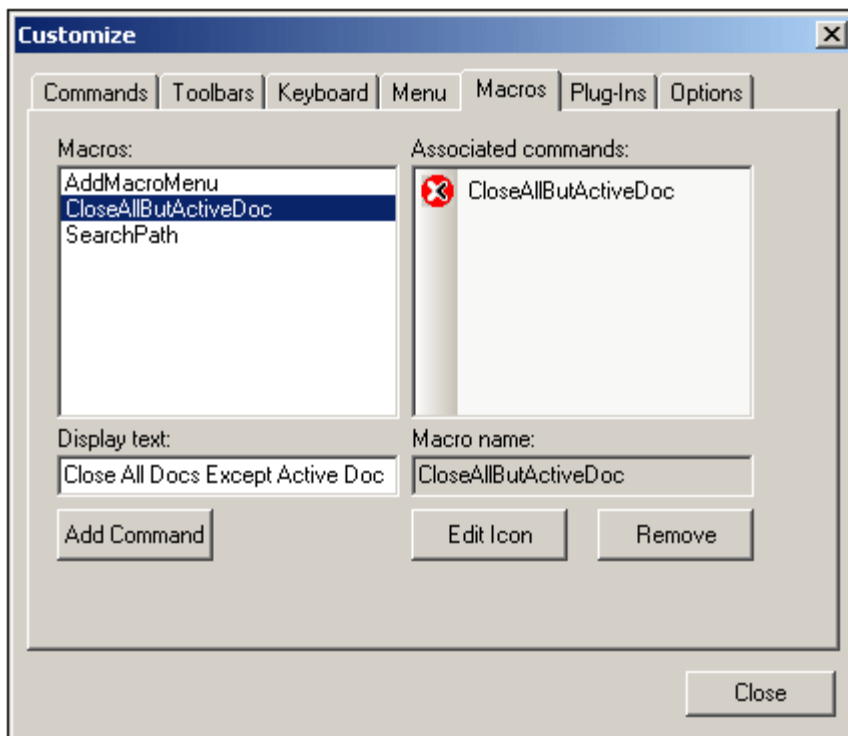


From the submenu of available macros, select the macro to run. The macro will be executed.

Toolbar icon

You can create an icon in the toolbar or a menu item that runs a selected macro. To do this,

click **Tools | Customize | Macros**. This causes the Customize dialog to be displayed (*screenshot below*).



Now do the following:

1. In the Macros tab of the Customize dialog, select the required macro from the Macros pane. The macros in the Macros pane are those in the active Scripting Project (which is specified in the Scripting tab of the Options dialog).
2. In the *Display Text* input field enter the name of the icon. This name will appear when the cursor is placed over the icon when it is in the toolbar.
3. Click **Add Command** to add it to the list of commands.
4. Select the command and click **Edit Icon** to create a new icon.
5. Drag the finished icon from the *Associated Commands* pane and drop it on to the toolbar or menu when the cursor changes from an arrow to an I-beam or line.
6. Macros can even be assigned their own shortcuts in the Keyboard tab of the Customize dialog (see *screenshot above*).

To remove the toolbar icon, open the Macros tab of the Customize dialog and drag the icon out of the toolbar and into the *Associated Commands* pane. Select the command in the *Associated Commands* pane and click **Remove** to remove the command from the pane.

Item in the Tools menu

The XMLSpy API includes a function, `AddMacroMenuItem()`, to add macros as menu items to the **Tools** menu. This function can be used to add one or more macros to the **Tools | Macros** list of macros. Typically, you should do this as follows:

1. Add the macro menu item by calling the XMLSpy API function, `AddMacroMenuItem()`.

```
Application.AddMacroMenuItem("DeleteElements","Delete Elements
Dialog");
```

- The function's first parameter (`DeleteElements` in the example listing above) is the

name of the macro. If you run the macro and there is an open project having scripts associated with it, XMLSpy searches for the macro in the project scripts first. If there are no project scripts, or if XMLSpy cannot find the macro, then it looks for the macro in the global scripts.

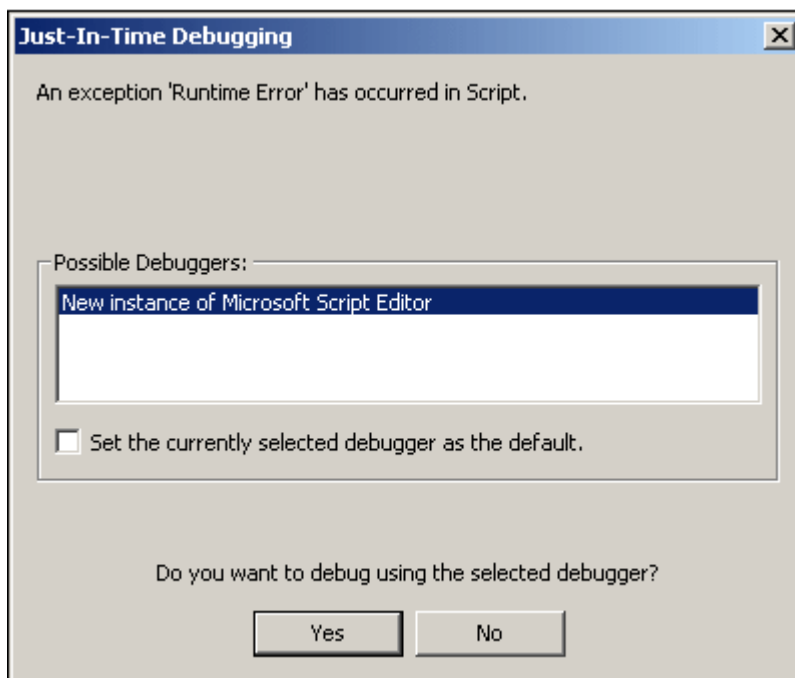
- The second parameter (`Delete Elements Dialog`) is the display text for the menu item.
2. Reset the **Tools** menu by calling `ClearMacroMenu()`. This removes all previously added menu items

The best way to call these two functions is with the `Autorun` macro of the global scripting project or the `On_OpenProject` event.

1.6.3 Debugging a Macro

You can debug a macro using an installed debugger. To do this, right-click the macro in the Scripting Project tree and select the command **Debug Macro**.

This pops up the Just-In-Time Debugging dialog (*screenshot below*), which lists the debuggers available on the machine. Select the debugger you wish to use and click **Yes**.



The selected debugger starts.

1.7 Programming Points

The following programming points should be noted:

- All namespaces and types of the following .NET assemblies can be accessed in the Microsoft .NET Framework per default:

```
System
System.Data
System.Design
System.Drawing
System.Windows.Forms
System.XML
```

Additional assemblies can be added to the scripting project via the [project's context menu](#), or dynamically (at runtime) in the source code by using [CLR.LoadAssembly](#).

- Out-parameters from methods of the XMLSpy API require special variables in JScript. Given below are some examples.

```
// use JScript method to access out-parameters
var strError = new Array(1);
var nErrorPos = new Array(1);
var objBadData = new Array(1);
bOK = objDoc.IsValid( strError, nErrorPos, objBadData ); END
```

- Out-parameters from methods of the .NET Framework require special variables in JScript.
For example:

```
var dictionary = CLR.Create( "System.Collections.Generic.Dictionary<
System.String,
System.String >" );
dictionary.Add( "1", "A" );
dictionary.Add( "2", "B" );

// use JScript method to access out-parameters
var strOut = new Array(1);
if ( dictionary.TryGetValue( "1", strOut ) ) // TryGetValue will set
the out parameter
    alert( strOut[0] ); // use out parameter
```

- .NET Methods that require integer arguments should not be called directly with JScript Number Objects which are Floating Point Values.

For example, instead of:

```
var objCustomColor = CLR.Static( "System.Drawing.Color" ). FromArgb(
128, 128, 128 );
```

use:

```
var objCustomColor = CLR.Static( "System.Drawing.Color" ). FromArgb(
Math.floor( 128 ), Math.floor( 128 ), Math.floor( 128 ) );
```

- To iterate .NET collections the JScript Enumerator as well as the .NET iterator technologies can be used:

For example:

```
// iterate using the JScript iterator
var itr = new Enumerator( coll );
for ( ; !itr.atEnd(); itr.moveNext() )
    alert( itr.item() );

// iterate using the .NET iterator
var itrNET = coll.GetEnumerator();
while( itrNET.MoveNext() )
    alert( itrNET.Current );
```

- .NET templates can be instantiated as shown below:

```
var coll = CLR.Create( "System.Collections.Generic.List<System.String>" );
```

or

```
CLR.Import( "System" );
CLR.Import( "System.Collections.Generic" );
var dictionary = CLR.Create( "Dictionary<String, Dictionary<String, String>>" );
```

- .NET Enum values are accessed as shown below:


```
var enumValStretch = CLR.Static( "System.Windows.Forms.ImageLayout" ).Stretch;
```
- Enumeration literals, as defined in the Altova type libraries, can now be used instead of numerical values.

```
objExportXMIFileDialog.XMIType = eXMI21ForUML23;
```

1.7.1 Built-in Commands

This section lists:

- [Built-in commands](#)
[alert](#)
[conform](#)
[doevents](#)
[createForm](#)
[lastform](#)
[prompt](#)
[showform](#)
[watchdog](#)
- [.NET interoperability](#) commands
[CLR.Create](#)
[CLR.Import](#)
[CLR.LoadAssembly](#)
[CLR.ShowImports](#)
[CLR.ShowLoadedAssemblies](#)
[CLR.Static](#)

Built-in commands

The following built-in commands are available.

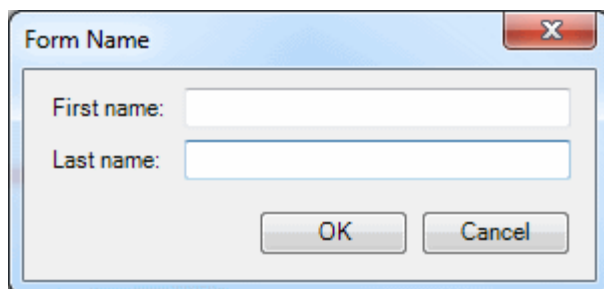
```
ShowForm(strFormName : String)
```

Instantiates a New Form object from the given form name and immediately shows it as Dialog.
Return Value: A Number that represents the generated DialogResult (`System.Windows.Forms.DialogResult`).

Example:

```
var dialogResult = ShowForm( "FormName" );
```

Shows Form "FormName" as Dialog:



The DialogResult can be evaluated e.g. by:

```
if ( dialogResult == CLR.Static( "System.Windows.Forms.DialogResult" ).  
OK )  
    alert( "ok" );  
else  
    alert( "cancel" );
```

CreateForm(strFormName : String)

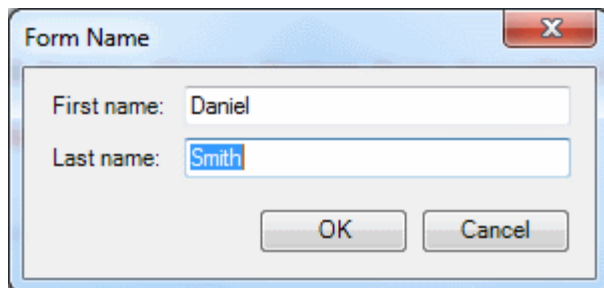
Instantiates a New Form object from the given Form name.

Return Value: The Form object (`System.Windows.Forms.Form`) of the given name, or null if no Form with such name exists.

Example:

```
var myForm = CreateForm( "FormName" );  
if ( myForm != null )  
{  
    myForm.textboxFirstName.Text = "Daniel";  
    myForm.textboxLastName.Text = "Smith";  
    var dialogResult = myForm.ShowDialog();  
}
```

Shows Form "FormName" as Dialog - TextBoxes are initialized:



The DialogResult can be evaluated e.g. by:

```
if ( dialogResult == CLR.Static( "System.Windows.Forms.DialogResult" ).  
OK )  
    alert( "ok" );  
else
```

```
alert( "cancel" );
```

lastform

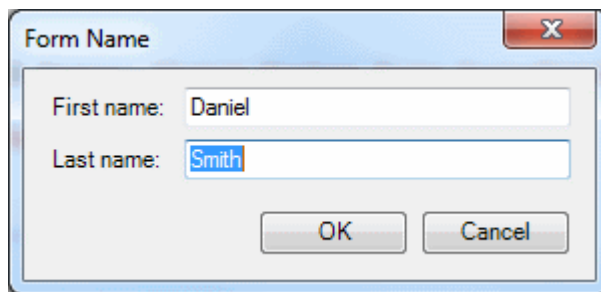
This global field can be used to conveniently access the last form object that was created.

Return Value: Returns a reference to the last form object (`System.Windows.Forms.Form`) that was successfully instantiated via `CreateForm()` or `ShowForm()`.

Example:

```
CreateForm( "FormName" );
if ( lastform != null )
{
    lastform.textBoxFirstName.Text = "Daniel";
    lastform.textBoxLastName.Text = "Smith";
    var dialogResult = lastform.ShowDialog();
}
```

Shows Form "FormName" as Dialog - TextBoxes are initialized (similar to the `CreateForm` example above):

**doevents()**

Processes all Windows messages currently in the message queue.

Return Value: None

Example:

```
for ( i=0; i < nLongLastingProcess; ++i )
{
    // do long lasting process

    doevents(); // process windows messages; give UI a chance to
update
}
```

watchdog(bEnable : boolean)

Long running CPU-intensive scripts cause the watchdog to ask the user if the script should be terminated. The `watchdog()` method is used to disable or enable this behavior.

Per default the watchdog is enabled.

Return Value: None

Example:

```
watchdog( false ); // disable watchdog - we know the next statement is
CPU intensive but it will terminate for sure
doCPUIntensiveScript();
watchdog( true ); // re-enable watchdog
```

Usage tip:

Calling `watchdog(true)` can also be used to reset the watchdog. This can be useful before executing long running (CPU intensive) tasks to ensure they have the maximum allowed script processing quota.

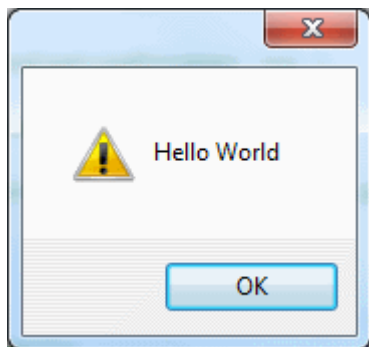
alert(strMessage : String) or MsgBox(strMessage : String)

An alert box is used to show a given message. The user will have to click "OK" to proceed.

Return Value: None

Example:

```
alert( "Hello World" );
```



confirm(strMessage : String)

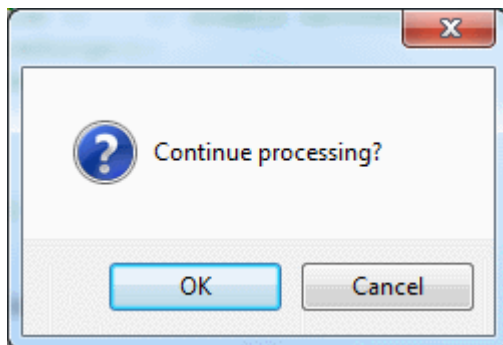
Opens a dialog that shows the given confirm message.

A confirm box is often used to verify or accept something. The user will have to click either "OK" or "Cancel" to proceed.

Return Value: A Boolean that represents the users answer. If the user clicks "OK", the dialog returns true, if the user clicks "Cancel", the dialog returns false.

Example:

```
if ( confirm( "Continue processing?" ) == false )  
    return;
```



prompt(strMessage : String, strDefault : String)

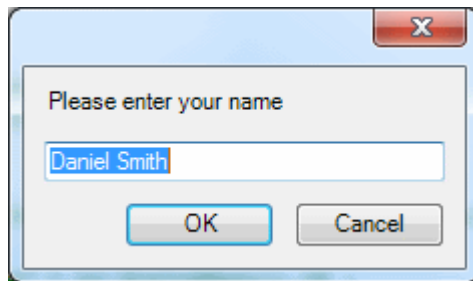
Opens a dialog that shows the given prompt message and a TextBox control with a default answer.

A prompt box is often used to input a simple string value.

Return Value: A String that contains the TextBox value or null if the user selected "Cancel".

Example:

```
var name = prompt( "Please enter your name", "Daniel Smith" );  
if ( name != null )  
    alert( "Hello " + name + "!" );
```

.NET interoperability commands

To allow further interoperability with the .NET Framework additional functions are provided under CLR.

CLR.Import(strNamespaceCLR : String)

This is the scripting equivalent to the C# *using* / VB.Net *imports* keyword. This allows to leave out the given namespaces in successive calls like `CLR.Create()` and `CLR.Static()`.

Return Value: None

Example:

Instead of always having to use full qualified names:

```
if ( ShowForm( "FormName" ) == CLR.Static(
"System.Windows.Forms.DialogResult" ).OK )
{
    var sName = lastform.textboxFirstName.Text + " " + lastform.
textboxLastName.Text;
    CLR.Static( "System.Windows.Forms.MessageBox" ).Show( "Hello " +
sName );
}
```

one can import namespaces and use the short form:

```
CLR.Import( "System.Windows.Forms" );
if ( ShowForm( "FormName" ) == CLR.Static( "DialogResult" ).OK )
{
    var sName = lastform.textboxFirstName.Text + " " + lastform.
textboxLastName.Text;
    CLR.Static( "MessageBox" ).Show( "Hello " + sName );
}
```

Please note:

Importing a namespace does not add or load the corresponding assembly to the scripting project!

Assemblies can be added to the scripting project by..... or dynamically (at runtime) in the source code by using [CLR.LoadAssembly](#).

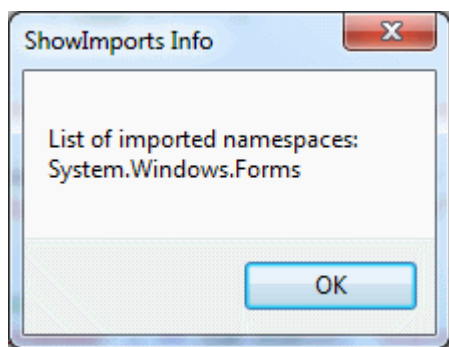
CLR.ShowImports()

Opens a MessageBox dialog that shows the currently imported namespaces. The user will have to click "OK" to proceed.

Return Value: None

Example:

```
CLR.ShowImports();
```



CLR.LoadAssembly(strAssemblyNameCLR : String)

Loads the .NET assembly with the given long assembly name or file path.

Return Value: A Boolean value. True if the assembly could be loaded, false otherwise.

Example:

```
// set clipboard text (if possible)
// System.Windows.Clipboard is part of the PresentationCore assembly,
// so load this assembly first:
if ( CLR.LoadAssembly( "PresentationCore, Version=3.0.0.0,
Culture=neutral, PublicKeyToken=31bf3856ad364e35", true ) )
{
    var clipboard = CLR.Static( "System.Windows.Clipboard" );
    if ( clipboard != null )
        clipboard.SetText( "HelloClipboard" );
}
```

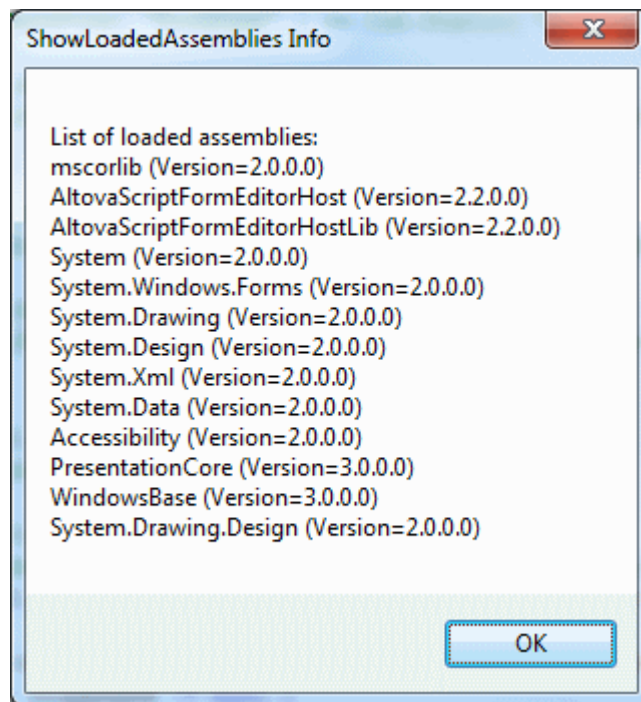
CLR.ShowLoadedAssemblies()

Opens a MessageBox dialog that shows the currently loaded assemblies. The user will have to click "OK" to proceed.

Return Value: None

Example:

```
CLR.ShowLoadedAssemblies();
```



CLR.Create(strTypeNameCLR : String, constructor arguments ...)

Creates a new .NET object instance for the given typename. If more than one argument is passed the successive arguments are interpreted as the arguments for the constructor of the .NET object.

Return Value: A reference to the created .NET object

Examples:

```
var objArray = CLR.Create("System.Collections.ArrayList");

var newItem = CLR.Create("System.Windows.Forms.ListViewItem",
    "NewItemText");

var coll = CLR.Create("System.Collections.Generic.List<System.String>");

CLR.Import("System");
CLR.Import("System.Collections.Generic");
var dictionary = CLR.Create("Dictionary<String, Dictionary<String,
String >>");
```

CLR.Static(strTypeNameCLR : String)

Gives access to .NET types that have no instances and contain only static members.

Return Value: A reference to the static .NET object

Examples:

```
var enumValStretch = CLR.Static("System.Windows.Forms.ImageLayout")
    .Stretch

var clipboard = CLR.Static("System.Windows.Clipboard");
clipboard.SetText("HelloClipboard");

if ( ShowForm("FormName") == CLR.Static(
    "System.Windows.Forms.DialogResult").OK )
    alert("ok");
```

```
else
    alert( "cancel" );
```

Form usage and commands

Form usage is as follows:

With Form objects, the Form Component Tree can be accessed naturally via field access:

For example, suppose there is a Form designed as follows:

```
MyForm
  ButtonPanel
    OkButton
    CancelButton
  TextEditor
  AxMediaPlayer1

TrayComponents:
  MyTimer
```

The Form can then be instantiated from script as:

```
var objForm = CreateForm( "MyForm" );
```

To access one its components the field access can be used:

```
objForm.ButtonPanel.OkButton.Enabled = false;
```

or

```
objForm.TextEditor.Text = "Hello World";
```

To access Tray Components use the following method on the Form object:

```
var objTrayComponent = <A form object>.GetTrayComponent( strComponentName
: String);
```

In our example to get a reference to the Timer Component to enable it use the following:

```
var objTimer = objForm.GetTrayComponent( "MyTimer" );
objTimer.Enabled = true;
```

For ActiveX Controls the underlying COM object can be accessed via the OCX property:

```
var ocx = lastform.AxMediaPlayer1.OCX; // get underlying COM object
ocx.enableContextMenu = true;
ocx.URL = "mms://apasf.apa.at/fm4_live_worldwide";
```

1.8 Migrating to Scripting Editor 2010 and Later

The Scripting Editor in XMLSpy from version 2010 onwards uses a different underlying technology than earlier versions used. Consequently, scripting projects that were created with versions of XMLSpy prior to version 2010 might need to be modified. The following points need to be noted.

- If a previous Scripting Projects (.prj file) is opened with the new Scripting Editor (version 2010 and later), the visual layout of Forms will be migrated as faithfully as possible and scripts will be copied as they are in the .prj file. You will then need to modify the scripts to be in accordance with the new technology used by the Scripting Editor, and which is described in this documentation.
- **TheView** object: The old Scripting Environment provided an artificial property named `TheView` that was only accessible from inside event handlers. It was used to access the Form that triggered the event (either directly or from one of its child controls). The new Scripting IDE does **not** provide this artificial property but instead provides the same functionality, and much more, with orthogonal [built-in scripting helper functions](#) combined with the power of the .NET framework.
- Since all event handlers in the new Scripting Environment get a sender object as a first parameter, the source that triggered the event is always available. By calling the .NET function `FindForm()` on the sender object one can access the Form object easily. Alternatively (if only one Form is involved) the built-in property `lastform` can be used. Note that the use of `lastform` is not constrained to event handlers (as was the case with `TheView`). It can be used everywhere in script code.

Given below is a list of methods and properties of the `TheView` object, each accompanied by an alternative mechanism offered by the new Scripting Environment.

Methods

The following methods were provided by the `TheView` object and must be migrated as explained:

`Cancel()`

In the new scripting environment the same can be achieved with: `lastform.Close(); // Use .NET Form.Close()`

`IsFormOpen(Name as String) as Boolean`

Since for .NET Forms there is a distinction between showing a Form and instantiating a Form, the previous concept does not directly translate. Instead the user can ask if a certain Form is currently shown. For example:

```
var objFormPencilSelector = CreateForm("PencilSelector");
var objFormColorSelector = CreateForm("ColorSelector");
...
// Anywhere in code ...

if(objFormColorSelector.Visible)
{
    ...
}
```

`FormFind(Name as String) as Object`

The new Scripting Environment allows you to instantiate more Forms of the same kind. In the old Scripting Environment each Form could only exist once (as a Singleton). Thus there is no equivalent of `FormFind()`. In the new Scripting Environment.

OpenDoc(File as String)

The same can be achieved with: `Application.OpenDocument( File as String )`

PumpData()

This corresponds to the built-in function `doevents()` which processes all Windows messages currently in the message queue.

RunClick(), RunInitialize(), RunTerminate()

There is no direct replacement for these methods. Call the corresponding handlers directly instead.

Properties

The following properties were provided by the `TheView` object and must be migrated as explained:

ToolTipText as String

To use tooltips in the new scripting environment, the .NET infrastructure can be used. This allows fine-grained control of tooltip behaviour (adjusting delays, when to show, etc). For example, to provide tooltips for a Form with two controls, the following code could be added to the Form's `Load` event handler:

```
//Occurs whenever the user loads the form.
function MyForm_Load( objSender, e_EventArgs )
{
    // Create the ToolTip and associate with the Form container.
    var toolTip = CLR.Create( "System.Windows.Forms.ToolTip" );

    // Set up the delays for the ToolTip.
    toolTip.AutoPopDelay = 3000;
    toolTip.InitialDelay = 1000;
    toolTip.ReshowDelay = 500;

    // Force the ToolTip text to be displayed whether or not
    // the form is active.
    toolTip.ShowAlways = true;

    // Set up the ToolTip text for several Controls.
    toolTip.SetToolTip( objSender.ProgressBar1,
        "Shows the progress of the operation" );
    toolTip.SetToolTip( objSender.Button1,
        "Click Button to start the processing" );
}
```

Color as Long

Since all Form/controls in the new Scripting Environment are .NET controls from the `System.Windows.Forms` namespace, the possibilities to modify colors, background image, fonts, and all other visual aspects are numerous. For example, every Visual Component has the properties `BackColor` and `ForeColor` to modify the visual appearance. The following handler could be used to change the color of a button at runtime:

```
function TestForm_Button1_Click( objSender, e_EventArgs )
{
    objSender.BackColor = CLR.Static( "System.Drawing.Color" ).SlateBlue;
}
```

Please refer to the .NET documentation to find out more about this topic:

<http://msdn.microsoft.com/en-us/library/system.windows.forms.aspx>

2 IDE Plugins

XMLSpy allows you to create your own IDE plug-ins and integrate them into XMLSpy.

Use plug-ins to:

- Configure your version of XMLSpy, add commands through menus, icons, buttons etc.
- React to events from XMLSpy.
- Run your specific code within XMLSpy with access to the complete XMLSpy API

XMLSpy expects your plug-in to implement the [IXMLSpyPlugIn](#) interface. See [ATL sample files](#) for an example using C++. The relevant files are in the following folder of your XMLSpy installation:

```
C:\Documents and Settings\<<username>\My Documents\Altova\XMLSpy2012  
\Examples\CppIDEPlugIn
```

For a sample using VisualBasic, look in the following folder of your XMLSpy installation:

```
C:\Documents and Settings\<<username>\My Documents\Altova\XMLSpy2012  
\Examples\XMLSpyPlugInActiveX
```


2.1 Registration of IDE Plugins

XMLSpy maintains a specific key in the Registry where it stores all registered IDE plug-ins:

HKEY_CURRENT_USER\Software\Altova\XML Spy\Plugins

All values of this key are treated as references to registered plug-ins and must conform to the following format:

Value name:	ProgID of the plug-in
Value type:	must be REG_SZ
Value data:	CLSID of the component

Each time the application starts the values of the "Plugins" key is scanned, and the registered plug-ins are loaded.

Register plug-in manually

To register a plug-in manually, use the "Customize" dialog box of the XMLSpy "Tools" menu. Use the "Add Plug-In..." button to specify the DLL that implements your plug-in. XMLSpy registers the DLL as a COM server and adds the corresponding entry in its "Plugins" key.

If you experience problems with manual registration you can check if the CLSID of your plug-in is correctly registered in the "Plugins" key. If this is not the case, the name of your plug-in DLL was probably not sufficiently unique. Use a different name or perform direct registration.

Register plug-in directly

A plug-in can be directly registered as an IDE plug-in by first registering the DLL and then adding the appropriate value to the "Plugins" key of XMLSpy during plug-in setup for example. The new plug-in will be activated the next time XMLSpy is launched.

2.2 ActiveX Controls

ActiveX controls are supported. Any IDE PlugIn which is also an ActiveX control will be displayed in a Dialog Control Bar. A sample PlugIn that is also an ActiveX control is included in the `XMLSpyPlugInActiveX` folder in the `Examples` folder of your application folder.

2.3 Configuration XML

The IDE plug-in allows you to change the user interface (UI) of XMLSpy. This is done by describing each separate modification using an XML data stream. The XML configuration is passed to XMLSpy using the [GetUIModifications](#) method of the `IXMLSpyPlugIn` interface.

The XML file containing the UI modifications for the IDE PlugIn, must have the following structure:

```
<ConfigurationData>
  <ImageFile>path To image file</ImageFile>
  <Modifications>
    <Modification>
      ...
    </Modification>
    ...
  </Modifications>
</ConfigurationData>
```

You can define icons or toolbar buttons for the new menu items which are added to the UI of XMLSpy by the plug-in. The path to the file containing the images is set using the **ImageFile** element. Each image must be 16 x 16 pixels using max. 256 colors. The image references must be arranged from left to right in a single (`<ImageFile>...`) line. The rightmost image index value, is zero.

The **Modifications** element can have any number of **Modification** child elements. Each **Modification** element defines a specific change to the standard UI of XMLSpy. Starting with version 4.3, it is also possible to remove UI elements from XMLSpy.

Structure of Modification elements

All Modification elements consist of the following two child elements:

```
<Modification>
  <Action>Type of action</Action>
  <UIElement Type="type of UI element">
    </UIElement>
  </Modification>
```

Valid values for the **Action** element are:

Add - to add the following UI element to XMLSpy
 Hide - to hide the following UI element in XMLSpy
 Remove - to remove the UI element from the "Commands" list box, in the customize dialog

You can combine values of the **Action** element e.g. "Hide Remove"

The **UIElement** element describes any new, or existing UI element for XMLSpy. Possible elements are currently: new toolbars, buttons, menus or menu items. The **type** attribute, defines which UI element is described by the XML element.

Common UIElement children

The ID and Name elements are valid for all different types of XML UIElement fragments. It is however possible, to ignore one of the values for a specific type of UIElement e.g. Name is ignored for a separator.

```
<ID></ID>
<Name></Name>
```

If **UIElement** describes an existing element of the UI, the value of the ID element is predefined

by XMLSpy. Normally these ID values are not known to the public. If the XML fragment describes a new part of the UI, then the ID is arbitrary and the value should be less than 1000. The **Name** element sets the textual value. Existing UI elements can be identified just by name, for e.g. menus and menu items with associated sub menus. For new UI elements, the **Name** element sets the caption e.g. the title of a toolbar, or text for a menu item.

Toolbars and Menus

To define a toolbar its necessary to specify the ID and/or the name of the toolbar. An existing toolbar can be specified using only the name, or by the ID if it is known. To create a **new** toolbar both values must be set. The **type** attribute must be equal to "ToolBar".

```
<UIElement Type="ToolBar">
  <ID>1</ID>
  <Name>TestPlugIn</Name>
</UIElement>
```

To specify an XMLSpy menu you need two parameters:

- The ID of the menu bar which contains the menu. If no XML documents are open in the main window, the menu bar ID is 128. If one or more XML documents are open, the menu bar ID is 129.
- The menu name. Menus do not have an associated ID value. The following example defines the "Edit" menu of the menu bar which is active, when at least one XML document is open:

```
<UIElement Type="Menu">
  <ID>129</ID>
  <Name>Edit</Name>
</UIElement>
```

An additional element is used if you want to create a new menu. The **Place** element defines the position of the new menu in the menu bar:

```
<UIElement Type="Menu">
  <ID>129</ID>
  <Name>PlugIn Menu</Name>
  <Place>12</Place>
</UIElement>
```

A value of -1 for the **Place** element sets the new button or menu item at the end of the menu or toolbar.

Commands

If you add a new command, through a toolbar button or a menu item, the **UIElement** fragment can contain any of these sub elements:

```
<MacroName></MacroName>
<Info></Info>
<ImageID></ImageID>
```

If **MacroName** is specified, XMLSpy searches for a macro with the same name in the scripting environment and executes it each time this command is processed. The **Info** element contains a short description string which is displayed in the status bar, when the mouse pointer is over the associated command (button or menu item). **ImageID** defines the index of the icon the external image file. Please note that all icons are stored in one image file.

To define a toolbar button create an **UIElement** with this structure:

```
<UIElement Type="ToolBarItem">
  <!--don't reuse local IDs even the commands do the same-->
  <ID>5</ID>
```

```

<Name>Open file from repository...</Name>
<!--Set Place To -1 If this is the first button To be inserted-->
<Place>-1</Place>
<ImageID>0</ImageID>
<ToolBarID>1</ToolBarID>
<!--instead of the toolbar ID the toolbar name could be used-->
<ToolBarName>TestPlugIn</ToolBarName>
</UIElement>

```

Additional elements to declare a toolbar button are **Place**, **ToolBarID** and **ToolBarName**. **ToolBarID** and **ToolBarName** are used to identify the toolbar which contains the new or existing button. The textual value of **ToolBarName** is case sensitive. The (UIElement) **type** attribute must equal "ToolBarItem".

To define a menu item, the elements **MenuID**, **Place** and **Parent** are available in addition to the standard elements used to declare a command. **MenuID** can be either 128 or 129. Please see "Toolbars and Menus" for more information on these values.

The **Parent** element is used to identify the **menu** where the new menu entry should be inserted. As sub menu items have no unique Windows ID, we need some other way to identify the parent of the menu item.

The value of the **Parent** element is a path to the menu item. The text value of the Parent element, must equal the **parent menu name** of the submenu, where the submenu name is separated by a colon. If the menu has no parent, because its not a submenu, add a colon to the beginning of the name. The **type** attribute must be set to "MenuItem". Example for an **UIElement** defining a menu item:

```

<UIElement Type="MenuItem">
  <!--the following element is a Local command ID-->
  <ID>3</ID>
  <Name>Open file from repository...</Name>
  <Place>-1</Place>
  <MenuID>129</MenuID>
  <Parent>: PlugIn Menu</Parent>
  <ImageID>0</ImageID>
</UIElement>

```

XMLSpy makes it possible to add toolbar separators and menus if the value of the **ID** element is set to 0.

2.4 ATL sample files

The following pages show how to create a simple XMLSpy IDE plug-in DLL using ATL. To build the DLL it is necessary to know about ATL, the wizards that generate new ATL objects, as well as MS VisualStudio.

To access the API the implementation imports the Type Library of XMLSpy. The code reads various properties and calls methods using the smart pointers provided by the `#import` statement.

In addition, the sample code uses the MFC class `CString` and the ATL conversion macros such as `W2T`.

At a glance the steps to create an ATL DLL are as follows:

1. Open VisualStudio and select "New..." from the "File" menu.
2. Select the "Projects" tab.
3. Select "ATL COM AppWizard" and type in a project name.
4. Select "Support for MFC" if you want to use MFC classes, or if you want to create a project for the sample code.

Having created the project files you can add an ATL object to implement the `IXMLSpyPlugIn` interface:

1. Select "New ATL Object..." from the "Insert" menu.
2. Select "Simple Object" from the wizard and click "Next".
3. Type in a name for the object.
4. On the "Attributes" tab, select "Custom" for the type of interface, and disable Aggregation.

These steps produce the skeleton code for the implementation of the IDE plug-in interface. Please see the following pages on how to modify the code and achieve some basic functionality.

2.4.1 Interface description (IDL)

The IDL of the newly created ATL object contains a declaration for one COM interface.

- This interface declaration must be replaced by the declaration of `IXMLSpyPlugIn` as shown below.
- The IDL must also contain the definition of the `SPYUpdateAction` enumeration.
- Replace the generated default interface name, (created by the wizard) with "`IXMLSpyPlugIn`" in the coclass declaration. The IDL should then look something like the example code below:

Having created the ATL object, you then need to implement the IDE plug-in interface of XMLSpy.

```
import "oaidl.idl";
import "ocidl.idl";

// ----- please insert the following block into your IDL file -----
typedef enum {
    spyEnable = 1,
    spyDisable = 2,
    spyCheck = 4,
    spyUncheck = 8
} SPYUpdateAction;
```

```

// ----- end insert block -----

// ----- E.g. Interface entry automatically generated by the ATL wizard -----
// [
//     object,
//     uuid( AB7CD86A-8145-429A-A1F3-270692E08AFC ),

//     helpstring("IXMLSpyPlugIn Interface")
//     pointer_default(unique)
// ]
// interface IXMLSpyPlugIn : IUnknown
// {
// };

// ----- end automatically generated Interface Entry

// ----- replace the Interface Entry (shown above) generated for you by the
// ATL wizard, with the following block -----

[
    odl,
    uuid(88F2A622-4B7E-42CD-8D04-3C0E5389DD85),
    helpstring("IXMLSpyPlugIn Interface")
]
interface IXMLSpyPlugIn : IUnknown
{
    HRESULT _stdcall OnCommand([in] long nID, [in] IDispatch* pXMLSpy);

    HRESULT _stdcall OnUpdateCommand([in] long nID, [in] IDispatch* pXMLSpy,
    [out, retval] SPYUpdateAction* pAction);

    HRESULT _stdcall OnEvent([in] long nEventID, [in] SAFEARRAY(VARIANT)*
    arrayParameters, [in] IDispatch* pXMLSpy, [out, retval] VARIANT*
    pReturnValue);

    HRESULT _stdcall GetUIModifications([out, retval] BSTR* pModificationsXML);

    HRESULT _stdcall GetDescription([out, retval] BSTR* pDescription);
};

// ----- end replace block -----

// ----- The code below is automatically generated by the ATL wizard and will
// look slightly different in your case -----

[
    uuid(24FE0D1B-3FC0-494E-B36E-1D4CE412B014),
    version(1.0),
    helpstring("XMLSpyIDEPlugInDLL 1.0 Type Library")
]
library XMLSPYIDEPLUGINDLLlib
{
    importlib("stdole32.tlb");
    importlib("stdole2.tlb");

    [
        uuid(3800E791-7F6B-4ACD-9E32-2AC184444501),
        helpstring("XMLSpyIDEPlugIn Class")
    ]
    coclass XMLSpyIDEPlugIn
    {
        [default] interface IXMLSpyPlugIn; // ----- define IXMLSpyPlugIn as the
        default interface -----
    }
}

```

```
};
};
```

2.4.2 Class definition

In the class definition of the ATL object, several changes must be made. The class has to derive from IXMLSpyPlugIn, the "Interface Map" needs an entry for IXMLSpyPlugIn, and the methods of the IDE plug-in interface must be declared:

```
#ifndef __XMLSPYIDEPLUGIN_H_
#define __XMLSPYIDEPLUGIN_H_

#include "resource.h"          // main symbols

////////////////////////////////////
// CXMLSpyIDEPlugIn
class ATL_NO_VTABLE CXMLSpyIDEPlugIn :
public CComObjectRootEx<CComSingleThreadModel>,
public CComCoClass<CXMLSpyIDEPlugIn, &CLSID_XMLSpyIDEPlugIn>,
public IXMLSpyPlugIn
{
public:
    CXMLSpyIDEPlugIn()
    {
    }

    DECLARE_REGISTRY_RESOURCEID(IDR_XMLSPYIDEPLUGIN)
    DECLARE_NOT_AGGREGATABLE(CXMLSpyIDEPlugIn)

    DECLARE_PROTECT_FINAL_CONSTRUCT()

    BEGIN_COM_MAP(CXMLSpyIDEPlugIn)
        COM_INTERFACE_ENTRY(IXMLSpyPlugIn)
    END_COM_MAP()

    // IXMLSpyIDEPlugIn
public:
    virtual HRESULT STDMETHODCALLTYPE OnCommand(long nID, IDispatch* pXMLSpy);

    virtual HRESULT STDMETHODCALLTYPE OnUpdateCommand(long nID, IDispatch* pXMLSpy,
        SPYUpdateAction* pAction);

    virtual HRESULT STDMETHODCALLTYPE OnEvent(long nEventID, SAFEARRAY** arrayParameters,
        IDispatch* pXMLSpy, VARIANT* pReturnValue);

    virtual HRESULT STDMETHODCALLTYPE GetUIModifications(BSTR* pModificationsXML);

    virtual HRESULT STDMETHODCALLTYPE GetDescription(BSTR* pDescription);
};

#endif // __XMLSPYIDEPLUGIN_H_
```

2.4.3 Implementation

The code below shows a simple implementation of an XMLSpy IDE plug-in. It adds a menu item and a separator (available with XMLSpy) to the Tools menu. Inside the OnUpdateCommand() method, the new command is only enabled when the active document is displayed using the Grid View. The command searches for the XML element which has the current focus, and opens any URL starting with "http://", from the textual value of the element.

```
////////////////////////////////////
// CXMLSpyIDEPlugIn
```



```

import "XMLSpy.tlb"
using namespace XMLSpyLib;

HRESULT CXMLSpyIDEPlugIn::OnCommand( long nID, IDispatch* pXMLSpy)
{
    USES_CONVERSION;

    if(nID == 1) {
        IApplicationPtr ipSpyApp;

        if( pXMLSpy) {
            if( SUCCEEDED( pXMLSpy->QueryInterface( __uuidof( IApplication), ( void
**) &ipSpyApp))) {
                IDocumentPtr ipDocPtr = ipSpyApp->ActiveDocument;

                // we assume that grid view is active
                if( ipDocPtr) {
                    IGridViewPtr ipGridPtr = ipDocPtr->GridView;

                    if( ipGridPtr) {
                        IXMLDataPtr ipXMLData = ipGridPtr->CurrentFocus;

                        CString strValue = W2T( ipXMLData->TextValue);

                        if( !strValue.IsEmpty() && (strValue.Left(7) == _T("http://")))
                            ::ShellExecute( NULL, _T("open")
, W2T( ipXMLData->TextValue), NULL, NULL, SW_SHOWNORMAL);
                    }
                }
            }
        }

        return S_OK;
    }

    HRESULT CXMLSpyIDEPlugIn::OnUpdateCommand( long nID, IDispatch* pXMLSpy,
    SPYUpdateAction* pAction)
    {
        *pAction = spyDisable;

        if(nID == 1) {
            IApplicationPtr ipSpyApp;

            if( pXMLSpy) {
                if( SUCCEEDED( pXMLSpy->QueryInterface( __uuidof( IApplication), ( void
**) &ipSpyApp))) {
                    IDocumentPtr ipDocPtr = ipSpyApp->ActiveDocument;

                    // only enable if grid view is active
                    if( (ipDocPtr != NULL) && (ipDocPtr->CurrentViewMode == spyViewGrid))
                        *pAction = spyEnable;
                }
            }
        }

        return S_OK;
    }

    HRESULT CXMLSpyIDEPlugIn::OnEvent( long nEventID, SAFEARRAY **arrayParameters,
    IDispatch* pXMLSpy, VARIANT* pReturnValue)
    {
        return S_OK;
    }

```

```

}

HRESULT CXMLSpyIDEPlugIn::GetUIModifications( BSTR* pModificationsXML)
{
    CComBSTR bstrMods = _T( " \
        <ConfigurationData> \
        <Modifications> " );
    // add "Open URL..." to Tools menu
    bstrMods.Append ( _T( " \
        <Modification> \
        <Action>Add</Action> \
        <UIElement type=\"MenuItem\"> \
        <ID>1</ID> \
        <Name>Open URL...</Name> \
        <Place>0</Place> \
        <MenuID>129</MenuID> \
        <Parent>: Tools</Parent> \
        </UIElement> \
        </Modification> " ));
    // add Seperator to Tools menu
    bstrMods.Append ( _T( " \
        <Modification> \
        <Action>Add</Action> \
        <UIElement type=\"MenuItem\"> \
        <ID>0</ID> \
        <Place>1</Place> \
        <MenuID>129</MenuID> \
        <Parent>: Tools</Parent> \
        </UIElement> \
        </Modification> " ));
    // finish modification description
    bstrMods.Append ( _T( " \
        </Modifications> \
        </ConfigurationData> " ));

    return bstrMods.CopyTo( pModificationsXML );
}

HRESULT CXMLSpyIDEPlugIn::GetDescription( BSTR* pDescription)
{
    CComBSTR bstrDescr = _T( "ATL C++ XMLSpy IDE PlugIn; This PlugIn demonstrates
the implementation of a simple ATL DLL as a IDE PlugIn for XMLSpy. " );
    return bstrDescr.CopyTo( pDescription );
}

```

2.5 IXMLSpyPlugIn

See also

Methods

[OnCommand](#)

[OnUpdateCommand](#)

[OnEvent](#)

[GetUIModifications](#)

[GetDescription](#)

Description

If a DLL is added to XMLSpy as an IDE plug-in, it is necessary that it registers a COM component that answers to an IXMLSpyPlugIn interface with the reserved uuid(88F2A622-4B7E-42CD-8D04-3C0E5389DD85), for it to be recognized as a plug-in.

2.5.1 OnCommand

See also

Declaration: `OnCommand( nID as long, pXMLSpy as IDispatch)`

Description

The `OnCommand()` method of the interface implementation, is called each time a command added by the the IDE plug-in (menu item or toolbar button) is processed. `nID` stores the command ID defined by the `ID` element of the respective `UIElement`.

`pXMLSpy` holds a reference to the dispatch interface of the `Application` object of XMLSpy.

Example

```
Public Sub IXMLSpyPlugIn_OnCommand( ByVal nID As Long, ByVal pXMLSpy As Object)
    If (Not (pXMLSpy Is Nothing)) Then
        Dim objDlg
        Dim objDoc As XMLSpyLib.Document
        Dim objSpy As XMLSpyLib.Application
        Set objSpy = pXMLSpy

        If nID = 3 Or nID = 5 Then
            Set objDlg = CreateObject("MSComDlg.CommonDialog")
            objDlg.Filter = "XML Files (*.xml)|*.xml| All Files (*.*)|*.*|| "
            objDlg.FilterIndex = 1
            objDlg.ShowOpen

            If Len(objDlg.FileName) > 0 Then
                Set objDoc = objSpy.Documents.OpenFile(objDlg.FileName, False)
                Set objDoc = Nothing
            End If
        End If

        If nID = 4 Or nID = 6 Then
            Set objDlg = CreateObject("MSComDlg.CommonDialog")
            objDlg.Filter = "All Files (*.*)|*.*|| "
            objDlg.Flags = cdlOFNPathMustExist
            objDlg.ShowSave

            If Len(objDlg.FileName) > 0 Then
                Set objDoc = objSpy.ActiveDocument

                If Not (objDoc Is Nothing) Then
                    objDoc.SetPathName objDlg.FileName
                End If
            End If
        End If
    End If
End Sub
```

```

        objDoc.Save
        Set objDoc = Nothing
    End If
End If
End If

    Set objSpy = Nothing
End If
End Sub

```

2.5.2 OnUpdateCommand

See also

Declaration: `OnUpdateCommand( nID as long, pXMLSpy as IDispatch) as SPYUpdateAction`

Description

The `OnUpdateCommand( )` method is called each time the visible state of a button or menu item needs to be set. `nID` stores the command ID defined by the `ID` element of the respective `UIElement`.

`pXMLSpy` holds a reference to the dispatch interface of the `Application` object.

Possible return values to set the update state are:

<code>spyEnable</code>	= 1
<code>spyDisable</code>	= 2
<code>spyCheck</code>	= 4
<code>spyUncheck</code>	= 8

Example

```

Public Function IXMLSpyPlugIn_OnUpdateCommand( ByVal nID As Long, ByVal
pXMLSpy As Object) As SPYUpdateAction
    IXMLSpyPlugIn_OnUpdateCommand = spyDisable

    If ( Not ( pXMLSpy Is Nothing) ) Then
        Dim objSpy As XMLSpyLib.Application
        Set objSpy = pXMLSpy

        If nID = 3 Or nID = 5 Then
            IXMLSpyPlugIn_OnUpdateCommand = spyEnable
        End If
        If nID = 4 Or nID = 6 Then
            If objSpy.Documents.Count > 0 Then
                IXMLSpyPlugIn_OnUpdateCommand = spyEnable
            Else
                IXMLSpyPlugIn_OnUpdateCommand = spyDisable
            End If
        End If
    End If
End Function

```

2.5.3 OnEvent

See also

Declaration: `OnEvent( nEventID as long, arrayParameters as SAFEARRAY( VARIANT) , pXMLSpy as IDispatch) as VARIANT`

Description

`OnEvent()` is called each time an event is raised from XMLSpy.

Possible values for `nEventID` are:

<code>On_BeforeStartEditing</code>	<code>= 1</code>
<code>On_EditingFinished</code>	<code>= 2</code>
<code>On_FocusChanged</code>	<code>= 3</code>
<code>On_Beforedrag</code>	<code>= 4</code>
<code>On_BeforeDrop</code>	<code>= 5</code>
<code>On_OpenProject</code>	<code>= 6</code>
<code>On_OpenDocument</code>	<code>= 7</code>
<code>On_CloseDocument</code>	<code>= 8</code>
<code>On_SaveDocument</code>	<code>= 9</code>

Events available since XMLSpy 4r4:

<code>On_DocEditDragOver</code>	<code>= 10</code>
<code>On_DocEditDrop</code>	<code>= 11</code>
<code>On_DocEditKeyDown</code>	<code>= 12</code>
<code>On_DocEditKeyUp</code>	<code>= 13</code>
<code>On_DocEditKeyPressed</code>	<code>= 14</code>
<code>On_DocEditMouseMove</code>	<code>= 15</code>
<code>On_DocEditButtonUp</code>	<code>= 16</code>
<code>On_DocEditButtonDown</code>	<code>= 17</code>
<code>On_DocEditContextMenu</code>	<code>= 18</code>
<code>On_DocEditPaste</code>	<code>= 19</code>
<code>On_DocEditCut</code>	<code>= 20</code>
<code>On_DocEditCopy</code>	<code>= 21</code>
<code>On_DocEditClear</code>	<code>= 22</code>
<code>On_DocEditSelectionChanged</code>	<code>= 23</code>

Events available since XMLSpy 2004:

<code>On_DocEditDragOver</code>	<code>= 10</code>

Events available since XMLSpy 2004r4 (type library version 1.4):

On_BeforeOpenProject	= 25
On_BeforeOpenDocument	= 26
On_BeforeSaveDocument	= 27
On_BeforeCloseDocument	= 28
On_ViewActivation	= 29
On_DocEditKeyboardEvent	= 30
On_DocEditMouseEvent	= 31

Events available since XMLSpy 2006 SP1 (type library version 1.5):

On_BeforeValidate	= 32

Events available since XMLSpy 2007 (type library version 1.6):

On_BeforeShowSuggestions	= 33
On_ProjectOpened	= 34
On_Char	= 35

Events available since XMLSpy 2009 (type library version 2.2):

On_Initialize	= 36
On_Running	= 37
On_Shutdown	= 38

Events available since XMLSpy 2012 (type library version 2.8):

On_AuthenticBeforeSave	= 39
On_AuthenticContextMenuActivated	= 40
On_AuthenticLoad	= 41
On_AuthenticToolBarButtonClicked	= 42
On_AuthenticToolBarButtonExecuted	= 43
On_AuthenticUserAddedXMLNode	= 44

The names of the events are the same as they appear in the Scripting Environment of XMLSpy. For IDE plug-ins the names used are immaterial. The events are identified using the `ID` value.

arrayParameters is an array which is filled with the parameters of the currently raised event. Order, type and meaning of the single parameters are available through the scripting environment of XMLSpy. The events module of a scripting project, contains predefined functions for all events prior to version 4.4. The parameters passed to the predefined functions are identical to the array elements of the `arrayParameters` parameter.

Events raised from the Authentic View of XMLSpy do not pass any parameters directly. An "event" object is used instead. The event object can be accessed through the Document object of the active document.

pXMLSpy holds a reference to the dispatch interface of the `Application` object of XMLSpy.

If the return value of `OnEvent()` is set, then neither the IDE plug-in, nor an event handler inside of the scripting environment will get this event afterwards. Please note that all IDE plug-ins get/process the event before the Scripting Environment does.

2.5.4 GetUIModifications

See also

Declaration: `GetUIModifications()` as `String`

Description

The `GetUIModifications()` method is called during initialization of the plug-in, to get the configuration XML data that defines the changes to the UI of XMLSpy. The method is called when the plug-in is loaded for the first time, and at every start of XMLSpy.

See also [Configuration XML](#) for a detailed description how to change the UI.

Example

```
Public Function IXMLSpyPlugIn_GetUIModifications() As String
    ' GetUIModifications() gets the XML file with the specified modifications
    of
    ' the UI from the config.xml file in the plug-in folder
    Dim strPath As String
    strPath = App.Path

    If Len(strPath) > 0 Then
        Dim fso As New FileSystemObject
        Dim file As file

        Set file = fso.GetFile(strPath & "\config.xml")

        If (Not (file Is Nothing)) Then
            Dim stream As TextStream
            Set stream = file.OpenAsTextStream(ForReading)

            ' this replaces the token '**path**' from the XML file with
            ' the actual installation path of the plug-in to get the image
file
            Dim strMods As String
            strMods = stream.ReadAll
            strMods = Replace(strMods, "**path**", strPath)

            IXMLSpyPlugIn_GetUIModifications = strMods
        Else
            IXMLSpyPlugIn_GetUIModifications = ""
        End If
    End If
End Function
```

2.5.5 GetDescription

See also

Declaration: `GetDescription()` as `String`

Description

`GetDescription()` is used to define the description string for the plug-in entries visible in the Customize dialog box.

Example

```
Public Function IXMLSpyPlugIn_GetDescription() As String
    IXMLSpyPlugIn_GetDescription = "Sample Plug-in for XMLSpy; This Plug-in
demonstrates the implementation of a simple VisualBasic DLL as a Plug-in for
XMLSpy."
```

```
End Function
```


3 Application API

The COM-based API of XMLSpy (also called the Application API from now on) enables other applications to use the functionality of XMLSpy. As a result, it is possible to automate a wide range of tasks, from validating an XML file to modifying complex XML content (with the [XMLData](#) interface).

XMLSpy and its Application API follow the common specifications for automation servers set out by Microsoft. It is possible to access the methods and properties of the Application API from common development environments, such as those using C, C++, VisualBasic, and Delphi, and with scripting languages like JScript and VBScript.

Execution environments for the Application API

The Application API can be accessed from the following execution environments:

- External programs (described [below](#) and in the [Overview](#) part of this section)
- From within the built-in Scripting Editor of XMLSpy. For a description of the scripting environment, see the section, [Scripting Editor](#).
- XMLSpy allows you to create and integrate your own plug-ins into the application using a special interface for plug-ins. A description of how to create plug-ins is given in the section [IDE Plug-ins](#).
- Via an ActiveX Control, which is available if the [integration package](#) is installed. For more information, see the section [ActiveX Integration](#).

External programs

In the [Overview](#) part of this section, we describe how the functionality of XMLSpy can be accessed and automated from external programs.

Using the Application API from outside XMLSpy requires an instance of XMLSpy to be started first. How this is done depends on the programming language used. See the section, [Programming Languages](#), for information about individual languages.

Essentially, XMLSpy will be started via its COM registration. Then the `Application` object associated with the XMLSpy instance is returned. Depending on the COM settings, an object associated with an already running XMLSpy can be returned. Any programming language that supports creation and invocation of COM objects can be used. The most common of these are listed below.

- [JScript](#) and [VBScript](#) script files have a simple syntax and are designed to access COM objects. They can be run directly from a DOS command line or with a double click on Windows Explorer. They are best used for simple automation tasks.
- [C#](#) is a full-fledged programming language that has a wide range of existing functionality. Access to COM objects can be automatically wrapped using C#.
- C++ provides direct control over COM access but requires relatively larger amounts of code than the other languages.
- [Java](#): Altova products come with native Java classes that wrap the Application API and provide a full Java look-and-feel.
- Other programming languages that make useful alternatives are: Visual Basic for Applications, Perl, and Python.

Programming points

The following limitations must be considered in your client code:

- Be aware that if your client code crashes, instances of XMLSpy may still remain in the system.
- Don't hold references to objects in memory longer than you need them, especially those from the `XMLData` interface. If the user interacts between two calls of your client, then there is no guarantee that these references are still valid.
- Don't forget to disable dialogs if the user interface is not visible.
- See [Error handling in JScript](#) (and in [C#](#) and [Java](#)) for details of how to avoid annoying error messages.
- Free references explicitly if you are using C or C++.

This documentation

This documentation section about the Application API is broadly divided into two parts.

- The first part consists of an [Overview](#), which describes the object model for the API and explains how the API is accessed via various [programming languages](#).
- The second part is a reference section ([Interfaces](#) and [Enumerations](#)) that contains descriptions of the interface objects of the Application API.

3.1 Overview

This overview of the Application API is organized as follows:

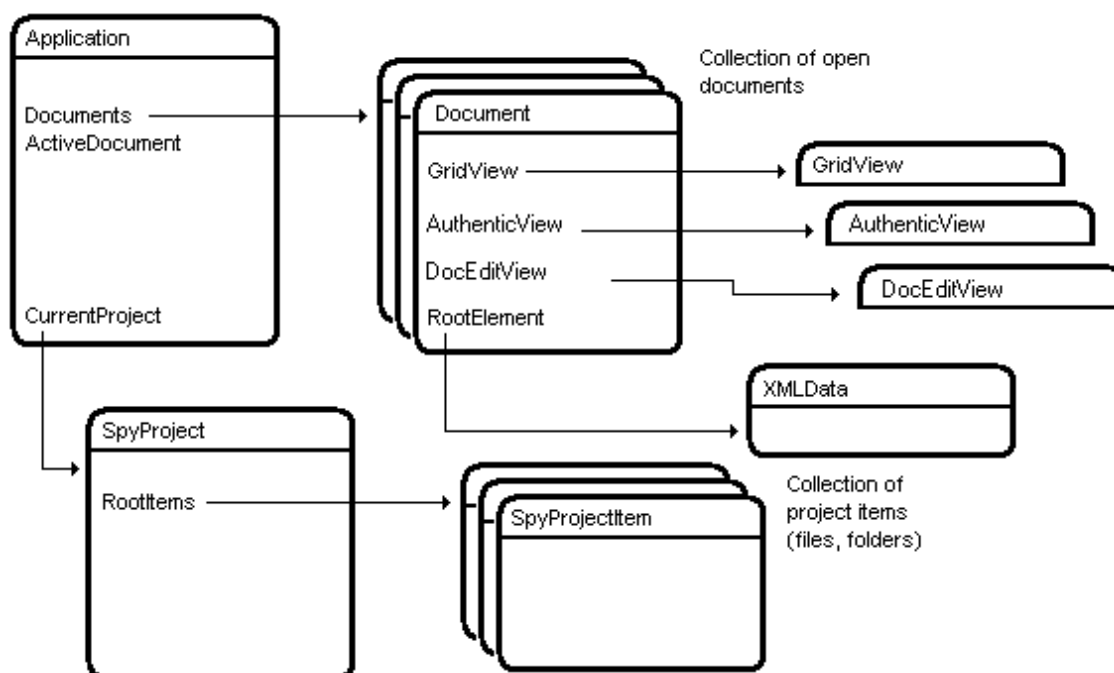
- [The Object Model](#) describes the relationships between the objects of the Application API.
- [Programming Languages](#) explains how the most commonly used programming languages (JScript, VBScript, C#, and Java) can be used to access the functionality of the Application API. Code listings from the example files supplied with your application package are used to describe basic mechanisms.
- [The DOM and XMLData](#) explains the relationship between the Application API's `XMLData` interface and the DOM.
- [Obsolete: Authentic View Row Operations](#) supplies information about obsolete objects for Authentic View table row operations.
- [Obsolete: Authentic View Editing Operations](#) supplies information about obsolete objects for Authentic View editing operations.

3.1.1 Object Model

The starting point for every application which uses the Application API is the [Application](#) object. This object contains general methods like import/export support and references to the open documents and any open project.

To create an instance of the `Application` object, call `CreateObject("XMLSpy.Application")` from VisualBasic or a similar function from your preferred development environment to create a COM object. There is no need to create any other objects in order to use the complete Application API; it is in fact not even possible. All other interfaces are accessed through other objects with the `Application` object as the starting point.

The picture below shows the links between the main objects of the Application API:



The application object consists of the following parts:

1. Document collection and reference to the active document.
2. Reference to current project and methods for creating and opening projects.
3. Methods to support the export to and import from databases, text files, and Word documents.
4. URL management.
5. Methods for macro menu items.

Once you have created an `Application` object you can start using the functionality of XMLSpy. In most cases, you either open a project and access the documents from there or you directly open a document via the [Documents](#) interface.

3.1.2 Programming Languages

Programming languages differ in the way they support COM access. A few examples for the most frequently used languages (*links below*) will help you get started. The code listings in this section show how basic functionality can be accessed. This basic functionality is included in the files in the API Examples folder and can be tested straight away. The path to the API Examples folder is given below:

Windows XP	C:/Documents and Settings/<username>/My Documents/ Altova/XMLSpy20XX/Examples/API/
Windows Vista, Windows 7	C:/Users/<username>/Documents/ Altova/XMLSpy20XX/Examples/API/

JScript

The JScript listings demonstrate the following basic functionality:

- [Start application or attach to a running instance](#)
- [Simple document access](#)
- [Iteration](#)
- [Error handling](#)
- [Events](#)
- [Import and export of data](#)

VBScript

VBScript is different than JScript only syntactically; otherwise it works in the same way. The listings below describe an example of how VBScript can be used. For more information, refer to the [JScript examples](#).

- [Events](#): Shows how events are handled using VBScript.

C#

C# can be used to access the Application API functionality. The code listings show how to access the API for certain basic functionality.

- [Start XMLSpy](#): Starts XMLSpy, which is registered as an automation server, or activates the program if XMLSpy is already running.
- [Open OrgChart.pxf](#): Locates one of the example documents installed with XMLSpy and opens it. If this document is already open it becomes the active document.

- [OnDocumentOpened Event On/Off](#): Shows how to listen to XMLSpy events. When turned on, a message box will pop up after a document has been opened.
- [Open ExpReport.xml](#): Opens another example document.
- [Toggle View Mode](#): Changes the view of all open documents between Text View and Authentic View. The code shows how to iterate through open documents.
- [Validate](#): Validates the active document and shows the result in a message box. The code shows how to handle errors and COM output parameters.
- [Shutdown XMLSpy](#): Stops XMLSpy.

Java

The XMLSpy API can be accessed from Java code. [This section](#) explains how some basic XMLSpy functionality can be accessed from Java code. It is organized into the following sub-sections:

- [Mapping Rules for the Java Wrapper](#)
- [Example Java Project](#)
- [Application Startup and Shutdown](#)
- [Simple Document Access](#)
- [Iterations](#)
- [Use of Out-Parameters](#)
- [Event Handlers](#)

JScript

This section contains listings of JScript code that demonstrate the following basic functionality:

- [Start application or attach to a running instance](#)
- [Simple document access](#)
- [Iteration](#)
- [Error handling](#)
- [Events](#)
- [Import and export of data](#)
- [Example: Bubble Sort Dynamic Tables](#)

Example files

The code listings in this section are available in example files that you can test as is or modify to suit your needs. The JScript example files are located in the `JScript` folder of the API Examples folder:

Windows XP	C:/Documents and Settings/<username>/My Documents/ Altova/XMLSpy20XX/Examples/API/
Windows Vista, Windows 7	C:/Users/<username>/Documents/ Altova/XMLSpy20XX/Examples/API/

The example files can be run in one of two ways:

- *From the command line*: Open a command prompt window and type the name of one of the example scripts (for example, `Start.js`). The Windows Scripting Host that is packaged with newer versions of Windows (XP, Vista, 7) will execute the script.
- *From Windows Explorer*: In Windows Explorer, browse for the JScript file and double-click it. The Windows Scripting Host that is packaged with newer versions of Windows (XP, Vista, 7) will execute the script. After the script is executed, the command console

gets closed automatically.

Start Application

The JScript below starts the application and shuts it down. If an instance of the application is already running, the running instance will be returned.

Script listing

The JScript listing below is explained with comments in the code.

```
// Initialize application's COM object. This will start a new instance of the
// application and
// return its main COM object. Depending on COM settings, the main COM object of an
// already
// running application might be returned.
try
{
    objSpy = WScript.GetObject("", "XMLSpy.Application");
}
catch(err)
{
    Exit("Can't access or create XMLSpy.Application");
}

// if newly started, the application will start without its UI visible. Set it to
// visible.
objSpy.Visible = true;

WScript.Echo("Hello");

objSpy.Visible = false;           // will shutdown application if it has no more COM
connections                      // will keep application running with UI visible
//objSpy.Visible = true;          // will keep application running with UI visible
```

Running the script

The JScript code listed above is available in the file `Start.js` located in the JScript folder of the API Examples folder:

Windows XP	C:/Documents and Settings/<username>/My Documents/ Altova/XMLSpy20XX/Examples/API/
Windows Vista, Windows 7	C:/Users/<username>/Documents/ Altova/XMLSpy20XX/Examples/API/

To run the script, start it from a command prompt window or from Windows Explorer.

Simple Document Access

The JScript listing below shows how to open documents, set a document as the active document, iterate through the open documents, and close documents.

Running the script

The JScript code listed below is available in the file `DocumentAccess.js` located in the JScript folder of the API Examples folder:

Windows XP	C:/Documents and Settings/<username>/My Documents/ Altova/XMLSpy20XX/Examples/API/
Windows Vista, Windows 7	C:/Users/<username>/Documents/ Altova/XMLSpy20XX/Examples/API/

To run the script, start it from a command prompt window or from Windows Explorer.

Script listing

The JScript listing below is explained with comments in the code.

```
// Initialize application's COM object. This will start a new instance of the
// application and
// return its main COM object. Depending on COM settings, a the main COM object of an
// already
// running application might be returned.
try
{
    objSpy = WScript.GetObject("", "XMLSpy.Application");
}
catch(err)
{
    Exit("Can't access or create XMLSpy.Application");
}

// if newly started, the application will start without its UI visible. Set it to
// visible.
objSpy.Visible = true;

// ***** code snippet for "Simple Document Access"
// *****

// Locate examples via USERPROFILE shell variable. The path needs to be adapted to major
// release versions.
objWshShell = WScript.CreateObject("WScript.Shell");
strExampleFolder = objWshShell.ExpandEnvironmentStrings("%USERPROFILE%") + "\\My
Documents\\Altova\\XMLSpy2012\\Examples\\";

// Tell XMLSpy to open two documents. No dialogs
objDoc1 = objSpy.Documents.OpenFile(strExampleFolder + "OrgChart.pxf", false);
objSpy.Documents.OpenFile(strExampleFolder + "ExpReport.xml", false);

// The document currently active can be easily located.
objDoc2 = objSpy.ActiveDocument;

// Let us make sure that the document is shown in grid view.
objDoc2.SwitchViewMode(0); // SPYViewModes.spyViewGrid = 0

// Now switch back to the document opened first
objDoc1.SetActiveDocument();

// ***** code snippet for "Simple Document Access"
// *****

// ***** code snippet for "Iteration"
// *****

// go through all open documents using a JScript Enumerator
bRequiresSaving = false;
for (var iterDocs = new Enumerator(objSpy.Documents); !iterDocs.atEnd(); iterDocs.
moveNext())
    if (iterDocs.item().IsModified)
        bRequiresSaving = true;

// go through all open documents using index-based access to the document collection
for (i = objSpy.Documents.Count; i > 0; i--)
```

```

objSpy.Documents.Item(i).Close(false);

// ***** code snippet for "Iteration"
// *****

//objSpy.Visible = false;           // will shutdown application if it has no more COM
connections
objSpy.Visible = true; // will keep application running with UI visible

```

Iteration

The JScript listing below shows how to iterate through the open documents.

Running the script

You can test this script by copying the listing below to a file, naming the file with a .js extension, and running the file from either the command line or Windows Explorer. You could copy the file to the JScript folder of the API Examples folder:

Windows XP	C:/Documents and Settings/<username>/My Documents/ Altova/XMLSpy20XX/Examples/API/
Windows Vista, Windows 7	C:/Users/<username>/Documents/ Altova/XMLSpy20XX/Examples/API/

To run the script, start it from a command prompt window or from Windows Explorer.

Script listing

The JScript listing below is explained with comments in the code.

```

// Initialize application's COM object. This will start a new instance of the
application and
// return its main COM object. Depending on COM settings, a the main COM object of an
already
// running application might be returned.
try
{
    objSpy = WScript.GetObject("", "XMLSpy.Application");
}
catch(err)
{
    Exit("Can't access or create XMLSpy.Application");
}

// if newly started, the application will start without its UI visible. Set it to
visible.
objSpy.Visible = true;

// ***** Accessing documents, setting the active document
// *****

// Locate examples via USERPROFILE shell variable. The path needs to be adapted to major
release versions.
objWshShell = WScript.CreateObject("WScript.Shell");
strExampleFolder = objWshShell.ExpandEnvironmentStrings("%USERPROFILE%") + "\\My
Documents\\Altova\\XMLSpy2012\\Examples\\";

// Tell XMLSpy to open two documents. No dialogs
objDoc1 = objSpy.Documents.OpenFile(strExampleFolder + "OrgChart.pxf", false);

```



```

objSpy.Documents.OpenFile(strExampleFolder + "ExpReport.xml", false);

// The document currently active can be easily located.
objDoc2 = objSpy.ActiveDocument;

// Let us make sure that the document is shown in grid view.
objDoc2.SwitchViewMode(0); // SPYViewModes.spyViewGrid = 0

// Now switch back to the document opened first
objDoc1.SetActiveDocument();

// ***** Two ways of iterating; close documents
// *****

// Go through all open documents using a JScript Enumerator
bRequiresSaving = false;
for (var iterDocs = new Enumerator(objSpy.Documents); !iterDocs.atEnd(); iterDocs.
moveNext())
    if (iterDocs.item().IsModified)
        bRequiresSaving = true;

// Go through all open documents using index-based access to the document collection
// and close each document. Comment out to leave documents open.

for (i = objSpy.Documents.Count; i > 0; i--)
    objSpy.Documents.Item(i).Close(false); // Closes the iterated documents

// ***** Shut down the application or not
// *****

//objSpy.Visible = false; // will shutdown application if it has no more COM
connections
objSpy.Visible = true; // will keep application running with UI visible

```

Error Handling

The Application API returns errors in two different ways:

- The `HRESULT` returned by every API method
- The `Info` interface of the Application API

Every API method returns an `HRESULT`. This return value gives the caller information about errors during execution of the method. If the call was successful, the return value is `S_OK`. The `HRESULT` option is commonly used in C/C++ programs.

However, programming languages such as VisualBasic and scripting languages (and other high-level development environments) don't give the programmer access to the `HRESULT` return of a COM call. Such languages use the `Info` interface, which is also supported by the Application API. If an error occurs, the Application API creates a new object that implements the `Info` interface. The information provided by the `Info` interface is imported by the development environment into its own error-handling mechanism.

JScript error handling

JScript provides a try-catch mechanism to deal with errors raised from COM calls. An error object containing the necessary information is declared. The listing below shows how.

```

// go through all open documents using a JScript Enumerator
bRequiresSaving = false;
for (var iterDocs = new Enumerator(objSpy.Documents); !iterDocs.atEnd(); iterDocs.

```

```

moveNext())
{
    if (iterDocs.item().IsModified)
        bRequiresSaving = true;

    var strErrorText = new Array(1);
    var nErrorNumber = new Array(1);
    var errorData = new Array(1);

    if (!iterDocs.item().IsValid(strErrorText, nErrorNumber, errorData))
    {
        var text = strErrorText;
        // access that XMLData object only if filled in
        if (errorData[0] != null)
            text += "(" + errorData[0].Name + "/" + errorData[0].TextValue +
    ");";

        WScript.Echo("Document \"" + iterDocs.item().Name + "\" validation error["
+ nErrorNumber + "]: " + text);
    }
    else
    {
        // The COM call succeeded and the document is valid.
        WScript.Echo("Document \"" + iterDocs.item().Name + "\" is valid.");
    }
}

```

Events

COM specifies that a client must register itself at a server for callbacks using the connection point mechanism. The automation interface for XMLSpy defines the necessary event interfaces. The way to connect to those events depends on the programming language you use in your client. The following code listing shows how this is done using JScript.

The method `WScript.ConnectObject` is used to receive events.

```

// The event-handler function
function DocEvent_OnBeforeCloseDocument(objDocument)
{
    WScript.Echo("Received event - before closing document");
}

// Create or connect to XMLSpy (or Authentic Desktop)
try
{
    // Create the environment and XMLSpy (or Authentic Desktop)
    objWshShell = WScript.CreateObject("WScript.Shell");
    objFSO = WScript.CreateObject("Scripting.FileSystemObject");
    objSpy = WScript.GetObject("", "XMLSpy.Application");

    // If only Authentic Desktop is installed (and XMLSpy is not installed) use:
    // objSpy = WScript.GetObject("", "AuthenticDesktop.Application")
}
catch(err)
{
    WScript.Echo ("Can't create WScript.Shell object or XMLSpy");
}

// Create document object and connect to its events
objSpy.Visible = true;
objDoc = objSpy.Documents.OpenFile ("C:\\Program
Files\\Altova\\XMLSpy2012\\Examples\\OrgChart.xml", false);
WScript.ConnectObject(objDoc, "DocEvent_");

// Keep running while waiting for the event
// In the meanwhile close this document in XMLSpy (or Authentic Desktop) manually
WScript.Echo ("Sleeping for 10 seconds ...");
WScript.Sleep (10000);

```

```
objDoc = null;  
WScript.Echo ("Stopped listening for event");  
objSpy.Quit();
```

Import and Export of Data

Before you implement your import and export tasks with the Application API, it is good practice to test the connections, parameters, SQL queries and so on in XMLSpy. In this way you are able to verify the results and make quick adjustments to import or export parameters.

Most of the methods for importing and exporting data are placed in the [Application](#) object, the remaining functions are accessible via the [Document](#) interface.

There is some preparatory work necessary, before the actual import or export can be started. Every import/export job consists of two parts. You need to define a connection to your data and the specific behaviour for the import/export process.

In case of an import, the connection is either a database, a text-file or a Word document. The behaviour is basically which data (columns) should be imported in XMLSpy.

In case of an export, the connection is either a database or a text file. Specify which data (elements of the XML file) and additional parameters (for example, automatic key generation or number of sub-levels) to use from the XML-structure.

The properties in the `DatabaseConnection`, `TextImportExportSettings` and `ExportSettings` interfaces have default values. See the corresponding descriptions in the [Interfaces](#) chapter for further information.

Descriptions

The sub-sections of this section describe each of these operations in detail.

- [Import from Database](#)
- [Export to Database](#)
- [Import from Text](#)
- [Export to Text](#)

Running the script

The JScript code listed below contains snippets for importing and exporting to database and text. It is available in the file `ImportExport.js` located in the `JScript` folder of the API Examples folder:

Windows XP	C:/Documents and Settings/<username>/My Documents/ Altova/XMLSpy20XX/Examples/API/
Windows Vista, Windows 7	C:/Users/<username>/Documents/ Altova/XMLSpy20XX/Examples/API/

To run the script, start it from a command prompt window or from Windows Explorer.

Script listing

The listing below contains code snippets for import and export to database and text.

```

// Initialize application's COM object. This will start a new instance of the
// application and
// return its main COM object. Depending on COM settings, a the main COM object of an
// already
// running application might be returned.
try
{
    objSpy = WScript.GetObject("", "XMLSpy.Application");
}
catch(err)
{
    Exit("Can't access or create XMLSpy.Application");
}

// if newly started, the application will start without its UI visible. Set it to
// visible.
objSpy.Visible = true;

// ***** Import from database: Start *****

// Locate examples via USERPROFILE shell variable. The path needs to be adapted to major
// release versions.
objWshShell = WScript.CreateObject("WScript.Shell");
strExampleFolder = objWshShell.ExpandEnvironmentStrings("%USERPROFILE%") + "\\My
Documents\\Altova\\XMLSpy2012\\Examples\\";

try
{
    // specify the source of data import
    objImpSettings = objSpy.GetDatabaseSettings();
    objImpSettings.File = strExampleFolder + "Tutorial\\Company.mdb";
    objImpSettings.SQLSelect = "SELECT * FROM Address";

    // column filter
    objElementList = objSpy.GetDatabaseImportElementList(objImpSettings);

    // import into a new XML file
    objImpDocFromDB = objSpy.ImportFromDatabase(objImpSettings, objElementList);
}
catch(err)
{
    WScript.Echo("Error importing from database.\n\n" +
        "Error: " + (err.number & 0xffff) + "\n" +
        "Description: " + err.description);
}

// ***** Import from database: End *****

// ***** Export to database: Start *****

//try
//{
    // set the behaviour of the export with ExportSettings
    objExpSettings = objSpy.GetExportSettings()

    //set the destination with DatabaseConnection
    objDB = objSpy.GetDatabaseSettings();
    objDB.CreateMissingTables = true;
    objDB.CreateNew = true;
    objDB.File = "C:\\Temp\\Export.mdb";

    try
    {
        objImpDocFromDB.ExportToDatabase(objImpDocFromDB.RootElement, objExpSettings,
objDB);
    }
}

```

```

catch(err)
{
    WScript.Echo("Error exporting to database.\n\n" +
        "Error: " + (err.number & 0xffff) + "\n" +
        "Description: " + err.description);
}

// ***** Export to database: End *****

// ***** Import from text: Start *****

try
{
    // specify the source of data import
    objImpSettings = objSpy.GetTextImportExportSettings();
    objImpSettings.ImportFile = strExampleFolder + "Tutorial\\Shapes.txt";
    objImpSettings.HeaderRow = false;

    // column filter
    objElementList = objSpy.GetTextImportElementList(objImpSettings);

    // import into a new XML file
    objImpDocFromText = objSpy.ImportFromText(objImpSettings,objElementList);
}
catch(err)
{
    WScript.Echo("Error importing from text file.\n\n" +
        "Error: " + (err.number & 0xffff) + "\n" +
        "Description: " + err.description);
}

// ***** Import from text: End *****

// ***** Export to text: Start *****

//try
//{
    objExpSettings = objSpy.GetExportSettings();
    objExpSettings.ElementList = objImpDocFromText.GetExportElementList(
objImpDocFromText.RootElement, objExpSettings);

    objTextExp = objSpy.GetTextImportExportSettings();
    objTextExp.HeaderRow = true;
    objTextExp.DestinationFolder = "C:\\Temp";
try
{
    objImpDocFromText.ExportToText(objImpDocFromText.RootElement, objExpSettings,
objTextExp);
}
catch(err)
{
    WScript.Echo("Error exporting to text.\n\n" +
        "Error: " + (err.number & 0xffff) + "\n" +
        "Description: " + err.description);
}

// ***** Export to text: End *****

//objSpy.Visible = false;           // will shutdown application if it has no more COM
connections
objSpy.Visible = true; // will keep application running with UI visible

```

Given below are the steps to establish a connection to an existing database for import:

1. Use a [DatabaseConnection](#) object and set the properties:
The method [Application.GetDatabaseSettings](#) returns a new object for a database connection:

```
objImpSettings = objSpy.GetDatabaseSettings();
```

You have to set either an **ADO connection string**,

- `objImpSettings.ADOConnection = strADOConnection`

or the **path** to an existing database file:

- `objImpSettings.File = strExampleFolder + "Tutorial\\Company.mdb";`

To complete the settings you create a **SQL select statement** to define the data to be queried:

- `objImpSettings.SQLSelect = "SELECT * FROM Address";`

2. Call [Application.GetDatabaseImportElementList](#) to get a collection of the resulting columns of the SQL query:

```
objElementList = objSpy.GetDatabaseImportElementList(objImpSettings);
```

This collection gives you the opportunity to control which columns should be imported and specify the datatype of the new elements. Each item of the collection represents one column to import. If you remove an item, the corresponding column will not be imported. You can additionally modify the [ElementListItem.ElementKind](#) property to set the datatype of the XML elements for each column.

Please consider that `GetDatabaseImportElementList()` executes the SQL query and could initiate a time-consuming call. To avoid this, it is possible to pass a null-pointer as the second parameter to `ImportFromDatabase()`; this imports all columns as plain XML elements.

3. Start the import with [Application.ImportFromDatabase](#):

```
objImpDocFromDB =  
objSpy.ImportFromDatabase(objImpSettings, objElementList);
```

Running the script

You can test the script below by copying the listing below to a file, naming the file with a `.js` extension, and running the file from either the command line or Windows Explorer. You could copy the file to the `JScript` folder of the API Examples folder:

Windows XP	C:/Documents and Settings/<username>/My Documents/ Altova/XMLSpy20XX/Examples/API/
Windows Vista, Windows 7	C:/Users/<username>/Documents/ Altova/XMLSpy20XX/Examples/API/

To run the script, start it from a command prompt window or from Windows Explorer.

Script listing

The following listing shows how data can be imported from a database.

```

// Initialize application's COM object. This will start a new instance of the
// application and
// return its main COM object. Depending on COM settings, a the main COM object of an
// already
// running application might be returned.
try
{
    objSpy = WScript.GetObject("", "XMLSpy.Application");
}
catch(err)
{
    Exit("Can't access or create XMLSpy.Application");
}

// if newly started, the application will start without its UI visible. Set it to
// visible.
objSpy.Visible = true;

// ***** Import from database *****

// Locate examples via USERPROFILE shell variable. The path needs to be adapted to major
// release versions.
objWshShell = WScript.CreateObject("WScript.Shell");
strExampleFolder = objWshShell.ExpandEnvironmentStrings("%USERPROFILE%") + "\\My
Documents\\Altova\\XMLSpy2012\\Examples\\";

try
{
    // specify the source of data import
    objImpSettings = objSpy.GetDatabaseSettings();
    objImpSettings.File = strExampleFolder + "Tutorial\\Company.mdb";
    objImpSettings.SQLSelect = "SELECT * FROM Address";

    // column filter
    objElementList = objSpy.GetDatabaseImportElementList(objImpSettings);

    // import into a new XML file
    objImpDocFromDB = objSpy.ImportFromDatabase(objImpSettings, objElementList);
}
catch(err)
{
    WScript.Echo("Error importing from database.\n\n" +
        "Error: " + (err.number & 0xffff) + "\n" +
        "Description: " + err.description);
}

// ***** End *****

//objSpy.Visible = false; // will shutdown application if it has no more COM
//connections
objSpy.Visible = true; // will keep application running with UI visible

```

To export data to a database, carry out the steps below:

1. Use a [DatabaseConnection](#) object and set the necessary properties.
All properties except `SQLSelect` are important for the export. `ADOConnection` or `File` defines the target for the output. You need to set only one of them.
2. Fill an [ExportSettings](#) object with the required values.
These properties are the same options as those available in the export dialog of XMLSpy. Select the menu option **Convert | Export to Text files/Database** to see the options and try a combination of export settings. After that it is easy to transfer these settings to the properties of the interface.

Call [Application.GetExportSettings](#) to get an `ExportSettings` object:

```
objExpSettings = objSpy.GetExportSettings()
```

3. Build an element list with [Document.GetExportElementList](#).
The element list enables you to eliminate XML elements from the export process. It also gives you information about the record and field count in the `RecordCount` and `FieldCount` properties. Set the [ExportSettings.ElementList](#) property to this collection. It is possible to set the element list to `null/Nothing` (default) to export all elements.
4. Call [Document.ExportToDatabase](#) to execute the export.
The description of the `ExportToDatabase` method contains also a code example for a database export.

Running the script

You can test the script below by copying the listing below to a file, naming the file with a `.js` extension, and running the file from either the command line or Windows Explorer. You could copy the file to the `JScript` folder of the API Examples folder:

Windows XP	C:/Documents and Settings/<username>/My Documents/ Altova/XMLSpy20XX/Examples/API/
Windows Vista, Windows 7	C:/Users/<username>/Documents/ Altova/XMLSpy20XX/Examples/API/

To run the script, start it from a command prompt window or from Windows Explorer.

Script listing

The following listing shows how data can be exported to a database.

```
// Initialize application's COM object. This will start a new instance of the
// application and
// return its main COM object. Depending on COM settings, a the main COM object of an
// already
// running application might be returned.
try
{
    objSpy = WScript.GetObject("", "XMLSpy.Application");
}
catch(err)
{
    Exit("Can't access or create XMLSpy.Application");
}

// if newly started, the application will start without its UI visible. Set it to
// visible.
objSpy.Visible = true;

// ***** Export to database *****
//try
//{
    // set the behaviour of the export with ExportSettings
    objExpSettings = objSpy.GetExportSettings()

    //set the destination with DatabaseConnection
```



```

objDB = objSpy.GetDatabaseSettings();
objDB.CreateMissingTables = true;
objDB.CreateNew = true;
objDB.File = "C:\\Temp\\Export.mdb";

try
{
    objImpDocFromDB.ExportToDatabase(objImpDocFromDB.RootElement, objExpSettings,
objDB);
}
catch(err)
{
    WScript.Echo("Error exporting to database.\n\n" +
        "Error: " + (err.number & 0xffff) + "\n" +
        "Description: " + err.description);
}

// ***** End *****

//objSpy.Visible = false;           // will shutdown application if it has no more COM
connections
objSpy.Visible = true; // will keep application running with UI visible

```

Importing data from a text file is similar to the import from a database. You must use other interfaces (described in steps 1 to 3 below) with different methods and properties:

1. Use a [TextImportExportSettings](#) object and set the properties:
The method [Application.GetTextImportExportSettings](#) returns a new object to specify a text file for import.

```
objImpSettings = objSpy.GetTextImportExportSettings();
```

You have to set at least the `ImportFile` property to the path of the file for the import. Another important property is `HeaderRow`. Set it to `False` if the text file does not contain a leading line as a header row.

```
objImpSettings.ImportFile = strExampleFolder + "Tutorial\\Shapes.txt";
```

2. Call [Application.GetTextImportElementList](#) to get a collection of all columns inside the text file:

```
objElementList = objSpy.GetTextImportElementList(objImpSettings);
```

3. Start the import with [Application.ImportFromText](#).

```
objImpDocFromText = objSpy.ImportFromText(objImpSettings,objElementList
);
```

Running the script

You can test the script below by copying the listing below to a file, naming the file with a `.js` extension, and running the file from either the command line or Windows Explorer. You could copy the file to the `JScript` folder of the API Examples folder:

Windows XP	C:/Documents and Settings/<username>/My Documents/ Altova/XMLSpy20XX/Examples/API/
Windows Vista, Windows 7	C:/Users/<username>/Documents/ Altova/XMLSpy20XX/Examples/API/

To run the script, start it from a command prompt window or from Windows Explorer.

Script listing

The following listing shows how data can be imported from text.

```
// Initialize application's COM object. This will start a new instance of the
// application and
// return its main COM object. Depending on COM settings, a the main COM object of an
// already
// running application might be returned.
try
{
    objSpy = WScript.GetObject("", "XMLSpy.Application");
}
catch(err)
{
    Exit("Can't access or create XMLSpy.Application");
}

// if newly started, the application will start without its UI visible. Set it to
// visible.
objSpy.Visible = true;

// ***** Import from text *****

try
{
    // specify the source of data import
    objImpSettings = objSpy.GetTextImportExportSettings();
    objImpSettings.ImportFile = strExampleFolder + "Tutorial\\Shapes.txt";
    objImpSettings.HeaderRow = false;

    // column filter
    objElementList = objSpy.GetTextImportElementList(objImpSettings);

    // import into a new XML file
    objImpDocFromText = objSpy.ImportFromText(objImpSettings, objElementList);
}
catch(err)
{
    WScript.Echo("Error importing from text file.\n\n" +
        "Error: " + (err.number & 0xffff) + "\n" +
        "Description: " + err.description);
}

// ***** End *****

//objSpy.Visible = false; // will shutdown application if it has no more COM
//connections
objSpy.Visible = true; // will keep application running with UI visible
```

To export data to text, carry out the steps below:

1. Use a [TextImportExportSettings](#) object and set the necessary properties.
2. Fill an [ExportSettings](#) object with the required values. See Item 2 in [Export to database](#).
3. Build an element list with [Document.GetExportElementList](#). See Item 3 in [Export to database](#).
4. Call [Document.ExportToText](#) to execute the export.

Running the script

You can test the script below by copying the listing below to a file, naming the file with a .js

extension, and running the file from either the command line or Windows Explorer. You could copy the file to the JScript folder of the API Examples folder:

Windows XP	C:/Documents and Settings/<username>/My Documents/ Altova/XMLSpy20XX/Examples/API/
Windows Vista, Windows 7	C:/Users/<username>/Documents/ Altova/XMLSpy20XX/Examples/API/

To run the script, start it from a command prompt window or from Windows Explorer.

Script listing

The following listing shows how data can be exported to text.

```
// Initialize application's COM object. This will start a new instance of the
// application and
// return its main COM object. Depending on COM settings, a the main COM object of an
// already
// running application might be returned.
try
{
    objSpy = WScript.GetObject("", "XMLSpy.Application");
}
catch(err)
{
    Exit("Can't access or create XMLSpy.Application");
}

// if newly started, the application will start without its UI visible. Set it to
// visible.
objSpy.Visible = true;

// ***** Export to text *****

//try
//{
    objExpSettings = objSpy.GetExportSettings();
    objExpSettings.ElementList = objImpDocFromText.GetExportElementList(
objImpDocFromText.RootElement, objExpSettings);

    objTextExp = objSpy.GetTextImportExportSettings();
    objTextExp.HeaderRow = true;
    objTextExp.DestinationFolder = "C:\\Temp";
try
{
    objImpDocFromText.ExportToText(objImpDocFromText.RootElement, objExpSettings,
objTextExp);
}
catch(err)
{
    WScript.Echo("Error exporting to text.\n\n" +
        "Error: " + (err.number & 0xffff) + "\n" +
        "Description: " + err.description);
}

// ***** End *****

//objSpy.Visible = false; // will shutdown application if it has no more COM
//connections
objSpy.Visible = true; // will keep application running with UI visible
```

Example: Bubble Sort Dynamic Tables

The following JScript snippet will sort any dynamic table by the table column identified by the current cursor position. The sort process is performed on screen. The undo buffer is available for all performed operations.

If you can run JScript on your computer - as you most likely will - copy the following code into a file with extension 'js'. Execute the script by double-clicking it in Windows Explorer, or run it from the command line.

```
// some useful XMLSpy enum constants
var spyAuthenticTag = 6;
var spyAuthenticTable = 9;
var spyAuthenticTableRow = 10;
var spyAuthenticTableColumn = 11;

var spyAuthenticRangeBegin = 2;

// example call for the sort table function
try
{
    var objSpy = WScript.GetObject("", "XMLSpy.Application");
    // If only Authentic is installed (and XMLSpy is not installed) use:
    // var objSpy = WScript.GetObject("", "AuthenticDesktop.Application")

    // use current selection to indicate which column to sort
    SortCurrentTable (objSpy.ActiveDocument.AuthenticView.Selection);
}
catch (err)
{
    WScript.Echo ("Please open a document in authentic view, and select a
        table column\n" +
        "Error : (" + (err.number & 0xffff) + ") " +
        err.description);
}

// we assume that XMLSpy is running, a document with a dynamic table
// is open, and the cursor is in a table column that will
// be used for sorting.
function SortCurrentTable (objCursor)
{
    if (objCursor.IsInDynamicTable())
    {
        // calculate current column index
        var nColIndex = 0;
        while (true)
        {
            // go left column-by-column
            try { objCursor.GotoPrevious(spyAuthenticTableColumn); }
            catch (err) { break; }
            nColIndex++;
        }

        // count number of table rows, so the bubble loops become simpler.
        // goto begin of table
        var objTableStart =
            objCursor.ExpandTo(spyAuthenticTable).CollapsToBegin().Clone();
        var nRows = 1;
        while (true)
        {
            // go down row-by-row
            try { objTableStart.GotoNext(spyAuthenticTableRow); }
            catch (err) { break; }
            nRows++;
        }
    }
}
```

```

// bubble sort through table
for ( var i = 0; i < nRows - 1; i++)
{
    // select correct column in first table row
    var objBubble =
    objCursor.ExpandTo(spyAuthenticTable).CollapsToBegin().Clone();
    objBubble.Goto (spyAuthenticTableColumn, nColIndex,
    spyAuthenticRangeBegin).ExpandTo(spyAuthenticTag);

    // bubble this row down as far as necessary
    for ( var j = 0; j < nRows - i - 1; j++)
    {
        var strField1 = objBubble.Text;
        // now look for the comparison table cell: start of next row and right
        // of the correct column
        var strField2 = objBubble.GotoNext(spyAuthenticTableRow).
            Goto (spyAuthenticTableColumn, nColIndex,
            spyAuthenticRangeBegin).
            ExpandTo(spyAuthenticTag).Text;
        if (strField1 > strField2)
        {
            objBubble.MoveRowUp(); // swap the rows
            // and re-calculate objBubble to select the cell to bubble
            objBubble.GotoNext(spyAuthenticTableRow).
                Goto (spyAuthenticTableColumn, nColIndex,
                spyAuthenticRangeBegin).
                ExpandTo(spyAuthenticTag);
        }
    }
}
}
else
    WScript.Echo ("please, select a table cell first");
}

```

VBScript

VBScript is syntactically different than JScript but works in the same way. This section contains a listing showing [how events are used with VBScript](#) and an [example](#).

For information about other functionality, refer to the JScript examples listed below:

- [Start application or attach to a running instance](#)
- [Simple document access](#)
- [Iteration](#)
- [Error handling](#)
- [Import and export of data](#)

Events

COM specifies that a client must register itself at a server for callbacks using the connection point mechanism. The automation interface for XMLSpy defines the necessary event interfaces. The way to connect to those events depends on the programming language you use in your client. The following code listing shows how this is done using VBScript.

The method `WScript.ConnectObject` is used to receive events.

```

' the event handler function
Function DocEvent_OnBeforeCloseDocument(objDocument)
    Call WScript.Echo("received event - before closing document")

```

```

End Function

' create or connect to XMLSPY
Set objWshShell = WScript.CreateObject("WScript.Shell")
Set objFSO = WScript.CreateObject("Scripting.FileSystemObject")
Set objSpy = WScript.GetObject("", "XMLSpy.Application")
' If only Authentic is installed (and XMLSpy is not installed) use:
' Set objSpy = WScript.GetObject("", "AuthenticDesktop.Application")

' create document object and connect to its events
objSpy.Visible = True
Set objDoc = objSpy.Documents.OpenFile ("C:\\Program
Files\\Altova\\XMLSPY2004\\Examples\\OrgChart.xml", False)
Call WScript.ConnectObject(objDoc, "DocEvent_")

' keep running while waiting on the event
' in the meantime close the document in XMLSPY manually
Call WScript.Echo ("sleeping for 10 seconds ...")
Call WScript.Sleep (10000)

Set objDoc = Nothing
Call WScript.Echo ("stopped listening for event")
Call objSpy.Quit

```

Example: Using Events

Authentic View supports event connection on a per-object basis. Implementation of this feature is based on COM connection points and is available in environments that support this mechanism.

The following example is a VBScript code example that shows how to use events from within a VBScript project.

```

Rem -----
Rem VBScript example that demonstrates how to use events.
Rem -----

' Event handler for OnSelectionChanged event of AuthenticView
Function AuthenticViewEvent_OnSelectionChanged( objAuthenticRange)
    If objAuthenticRange.FirstTextPosition <> objAuthenticRange.LastTextPosition Then
        Call WScript.Echo("Selection: " & objAuthenticRange.Text & vbNewLine & vbNewLine
& "Close this dialog.")
    Else
        Call WScript.Echo("Cursor position: " & objAuthenticRange.FirstTextPosition &
vbNewLine & vbNewLine & "Close this dialog.")
    End If
End Function

' Here starts the main code.
' Find out user's personal folder and locate one of the installed XMLSpy 2011 examples.
Set WshShell = WScript.CreateObject("WScript.Shell")
personalFolder = WshShell.ExpandEnvironmentStrings("%UserProfile%")
xmlspyExamplesFolder = personalFolder & "\\My Documents\\Altova\\XMLSpy2011\\Examples\\"
docPath = xmlspyExamplesFolder & "OrgChart.xml"

' Start/access XMLSpy and connect to its automation interface.
Set objSpy = GetObject("", "XMLSpy.Application")
' Make the UI of XMLSpy visible.
objSpy.Visible = True

' Create object to access windows file system and test if the our document exists.
Set fso = CreateObject("Scripting.FileSystemObject")
If fso.FileExists(docPath) Then
    ' open the document
    Call objSpy.Documents.OpenFile(docPath, False)
    set objDoc = objSpy.ActiveDocument

    ' switch active document to authentic view
    objDoc.SwitchViewMode 4 ' spyViewAuthentic

```

```

' Register for connection point events on the authentic view of the active document.
' Any function with a valid event name prefixed with "AuthenticViewEvent_" will
' be called when the corresponding event gets triggered on the specified object.
set objView = objDoc.AuthenticView
Call WScript.ConnectObject(objView, "AuthenticViewEvent_")
Call WScript.Echo("Events are connected." & vbNewLine & vbNewLine & "Now set or move
the cursor in XMLSpy." & vbNewLine & vbNewLine & "Close this dialog to shut down
XMLSpy.")

' To disconnect from the events delete the reference to the object.
set objView = Nothing
Else
Call WScript.Echo("The file " & docPath & " does not exist.")
End If

' shut down XMLSpy when this script ends
objSpy.Visible = False

```

C#

The C# programming language can be used to access the Application API functionality. You could use Visual Studio 2008 or Visual Studio 2010 to create the C# code, saving it in a Visual Studio project. Create the project as follows:

1. In Microsoft Visual Studio, add a new project using **File | New | Project**.
2. Add a reference to the XMLSpy Type Library by clicking **Project | Add Reference**. The Add Reference dialog pops up, displaying a list of installed COM components. Select the XMLSpy Type Library component from the list to add it.
3. Enter the code you want.
4. Compile the code and run it.

Example C# project

Your XMLSpy package contains an example C# project, which is located in the C# folder of the API Examples folder:

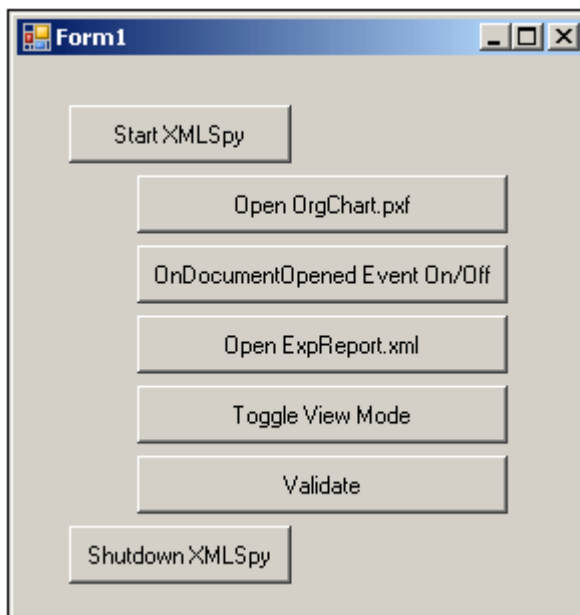
Windows XP	C:/Documents and Settings/<username>/My Documents/ Altova/XMLSpy20XX/Examples/API/
Windows Vista, Windows 7	C:/Users/<username>/Documents/ Altova/XMLSpy20XX/Examples/API/

You can compile and run the project from within Visual Studio 2008 or Visual Studio 2010.

The code listing below shows how basic application functionality can be used. This code is similar to the example C# project in the API Examples folder of your application package, but might differ slightly.

What the code listing below does

The example code listing below creates a simple user interface (*screenshot below*) with buttons that invoke basic XMLSpy operations:



- [Start XMLSpy](#): Starts XMLSpy, which is registered as an automation server, or activates the application if it is already running.
- [Open OrgChart.pxf](#): Locates one of the example documents installed with XMLSpy and opens it. If this document is already open it becomes the active document.
- [OnDocumentOpened Event On/Off](#): Shows how to listen to XMLSpy events. When turned on, a message box will pop up after a document has been opened.
- [Open ExpReport.xml](#): Opens another example document.
- [Toggle View Mode](#): Changes the view of all open documents between Text View and Authentic View. The code shows how to iterate through open documents.
- [Validate](#): Validates the active document and shows the result in a message box. The code shows how to handle errors and COM output parameters.
- [Shut down XMLSpy](#): Stops XMLSpy.

You can modify the code (of the code listing below or of the example C# project in the API Examples folder) in any way you like and run it.

Compiling and running the example

In the API Examples folder, double-click the file `AutomateXMLSpy_VS2008.sln` (to open it in Visual Studio 2008) or the file `AutomateXMLSpy_VS2010.sln` (to open it in Visual Studio 2010). Alternatively the file can be opened from within Visual Studio (with **File | Open | Project/Solution**). To compile and run the example, select **Debug | Start Debugging** or **Debug | Start Without Debugging**.

Code listing of the example

Given below is the C# code listing of the basic functionality of the form (`Form1.cs`) created in the `AutomateXMLSpy` example. Note that the code listed below might differ slightly from the code in the API Examples form. The listing below is commented for ease of understanding. Parts of the code are also presented separately in the sub-sections of this section, according to the Application API functionality they access.

The code essentially consists of a series of handlers for the buttons in the user interface shown

in the screenshot above.

```
namespace WindowsFormsApplication2
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();

            // An instance of XMLSpy accessed via its automation interface
            XMLSpyLib.Application XMLSpy;

            // Location of examples installed with XMLSpy
            String strExamplesFolder;

            private void Form1_Load(object sender, EventArgs e)
            {
                // Locate examples installed with XMLSpy
                // REMARK: You might need to adapt this if you have a different major
                version of the product
                strExamplesFolder = Environment.GetEnvironmentVariable("USERPROFILE") +
                "\\My Documents\\Altova\\XMLSpy2012\\Examples\\";
            }

            // Handler for the "Start XMLSpy" button
            private void StartXMLSpy_Click(object sender, EventArgs e)
            {
                if (XMLSpy == null)
                {
                    Cursor.Current = Cursors.WaitCursor;

                    // If no XMLSpy instance is open, create one and make it visible
                    XMLSpy = new XMLSpyLib.Application();
                    XMLSpy.Visible = true;

                    Cursor.Current = Cursors.Default;
                }
                else
                {
                    // If an instance of XMLSpy is already running, make sure it's visible
                    if (!XMLSpy.Visible)
                        XMLSpy.Visible = true;
                }
            }

            // Handler for the "Open OrgChart.pxf" button
            private void openOrgChart_Click(object sender, EventArgs e)
            {
                // Make sure there's a running XMLSpy instance, and that it's visible
                StartXMLSpy_Click(null, null);

                // Open one of the example files installed with the product
                XMLSpy.Documents.OpenFile(strExamplesFolder + "OrgChart.pxf", false);
            }

            // Handler for the "Open ExpReport.xml" button
            private void openExpReport_Click(object sender, EventArgs e)
            {
                // Make sure there's a running XMLSpy instance, and that it's visible
                StartXMLSpy_Click(null, null);

                // Open one of the sample files installed with the product.
                XMLSpy.Documents.OpenFile(strExamplesFolder + "ExpReport.xml", false);
            }

            // Handler for the "Toggle View Mode" button
            private void toggleView_Click(object sender, EventArgs e)
            {
                // Make sure there's a running XMLSpy instance, and that it's visible
                StartXMLSpy_Click(null, null);
            }
        }
    }
}
```

```

        // Iterate through all open documents and toggle view between Text View and
        Authentic View
        foreach (XMLSpyLib.Document doc in XMLSpy.Documents)
        {
            if (doc.CurrentViewMode == XMLSpyLib.SPYViewModes.spyViewText)
                doc.SwitchViewMode(XMLSpyLib.SPYViewModes.spyViewAuthentic);
            else
                doc.SwitchViewMode(XMLSpyLib.SPYViewModes.spyViewText);
        }

        // Handler for the "Shutdown XMLSpy" button
        // Shut down the application instance by explicitly releasing the COM object
        private void shutdownXMLSpy_Click(object sender, EventArgs e)
        {
            if (XMLSpy != null)
            {
                // Allow shutdown of XMLSpy by releasing the UI
                XMLSpy.Visible = false;

                // Explicitly release the COM object
                try
                {
                    while (System.Runtime.InteropServices.Marshal
                        .ReleaseComObject(XMLSpy) > 0) ;
                }
                finally
                {
                    // Disallow subsequent access to this object
                    XMLSpy = null;
                }
            }
        }

        // Handler for button "Validate"
        private void validate_Click(object sender, EventArgs e)
        {
            // COM errors are returned to C# as exceptions. We use a try/catch block to
            handle them
            try
            {
                // Method 'IsValid' is one of the few functions that uses output
                parameters
                // Use 'object' type for these parameters
                object strErrorText = "";
                object nErrorNumber = 0;
                object errorData = null;

                if (!XMLSpy.ActiveDocument.IsValid(ref strErrorText, ref nErrorNumber,
                    ref errorData))
                {
                    // The COM call succeeded but the document is not valid
                    // A detailed description of the problem is returned in
                    strErrorText, nErrorNumber and errorData
                    listBoxMessages.Items.Add("Document " + XMLSpy.ActiveDocument.Name
                        + " is not valid.");
                    listBoxMessages.Items.Add("\tErrorText : " + strErrorText);
                    listBoxMessages.Items.Add("\tErrorNumber: " + nErrorNumber);
                    listBoxMessages.Items.Add("\tElement : " + (errorData != null ?
                        ((XMLSpyLib.XMLData) errorData).TextValue : "null"));
                }
                else
                {
                    // The COM call succeeded and the document is valid
                    listBoxMessages.Items.Add("Document " + XMLSpy.ActiveDocument.Name
                        + " is valid.");
                }
            }
            catch (Exception ex)
            {
                // The COM call was not successful
                // Probably no application instance has been started or no document is
                open.
                listBoxMessages.Items.Add("Error validating active document: " +
                    ex.Message);
            }
        }
    }

```

```

// Event handler for OnDocumentOpened event
private void handleOnDocumentOpened( XMLSpyLib.Document i_ipDocument)
{
    MessageBox.Show("Document " + i_ipDocument.Name + " was opened!");
}

// Remember if the event handler is currently registered.
private bool bEventHandlerIsRegistered = false;

// Handler for button 'OnDocuemntOpened Event On/Off
private void toggleOnDocumentOpenedEvent_Click(object sender, EventArgs e)
{
    if (XMLSpy != null)
    {
        if (bEventHandlerIsRegistered)
            XMLSpy.OnDocumentOpened -= new XMLSpyLib.
_IApplicationEvents_OnDocumentOpenedEventHandler( handleOnDocumentOpened );
        else
            XMLSpy.OnDocumentOpened += new XMLSpyLib.
_IApplicationEvents_OnDocumentOpenedEventHandler( handleOnDocumentOpened );

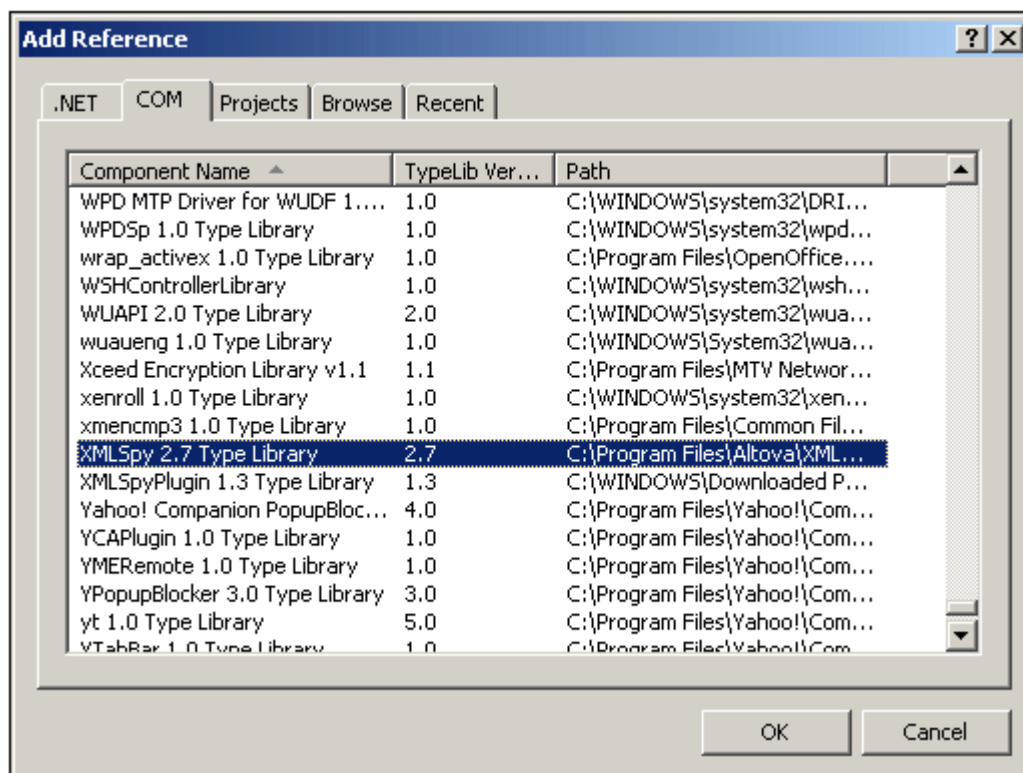
        bEventHandlerIsRegistered = !bEventHandlerIsRegistered;
    }
}
}
}

```

Add Reference to XMLSpy Application API

To add the application's type library as a reference in a .NET project, do the following:

1. With the .NET project open, click **Project | Add Reference**. The Add Reference dialog (screenshot below) pops up, displaying a list of installed COM components.



2. Select `XMLSpy X.Y Type Library` from the component list and finish.

Declare a variable to access the XMLSpy API

```
// An instance of XMLSpy is accessed via its automation interface.
XMLSpyLib.Application XMLSpy;
```

Application Startup and Shutdown

In the code snippets below, the methods `StartXMLSpy_Click` and `ShutdownXMLSpy_Click` are those assigned to buttons in the [AutomateXMLSpy example](#) that, respectively, start up and shut down the application. This example is located in the `C#` folder of the API Examples folder (see the file `Form1.cs`):

Windows XP	C:/Documents and Settings/<username>/My Documents/ Altova/XMLSpy20XX/Examples/API/
Windows Vista, Windows 7	C:/Users/<username>/Documents/ Altova/XMLSpy20XX/Examples/API/

You can compile and run the project from within Visual Studio 2008 or Visual Studio 2010.

Starting XMLSpy

The following code snippet from the [AutomateXMLSpy example](#) shows how to start up the application.

```
// Handler for the "Start XMLSpy" button
private void StartXMLSpy_Click(object sender, EventArgs e)
{
    if (XMLSpy == null)
    {
        Cursor.Current = Cursors.WaitCursor;

        // If no XMLSpy instance is running, we create one and make it visible
        XMLSpy = new XMLSpyLib.Application();
        XMLSpy.Visible = true;

        Cursor.Current = Cursors.Default;
    }
    else
    {
        // If an instance of XMLSpy is already running, make sure it's visible
        if (!XMLSpy.Visible)
            XMLSpy.Visible = true;
    }
}
```

Shutting down XMLSpy

The following code snippet from the [AutomateXMLSpy example](#) shows how to shut down the application.

```
// Handler for the "Shutdown XMLSpy" button
// Shut down the application instance by explicitly releasing the COM object
private void shutdownXMLSpy_Click(object sender, EventArgs e)
{
    if (XMLSpy != null)
    {

```

```

        // Allow shutdown of XMLSpy by releasing the UI
        XMLSpy.Visible = false;

        // Explicitly release COM object
        try
        {
            while (System.Runtime.InteropServices.Marshal
.ReleaseComObject( XMLSpy) > 0) ;
        }
        finally
        {
            // Disallow subsequent access to this object
            XMLSpy = null;
        }
    }
}

```

Opening Documents

The code snippets below (from the [AutomateXMLSpy example](#)) show how two files are opened via two separate methods assigned to two buttons in the user interface. Both methods use the same Application API access mechanism: [XMLSpy.Documents.OpenFile\(string, boolean\)](#).

The [AutomateXMLSpy example](#) (see the file *Form1.cs*) is located in the C# folder of the API Examples folder:

Windows XP	C:/Documents and Settings/<username>/My Documents/ Altova/XMLSpy20XX/Examples/API/
Windows Vista, Windows 7	C:/Users/<username>/Documents/ Altova/XMLSpy20XX/Examples/API/

You can compile and run the project from within Visual Studio 2008 or Visual Studio 2010.

Code snippet

```

// Handler for the "Open OrgChart.pxf" button
private void openOrgChart_Click(object sender, EventArgs e)
{
    // Make sure there's a running XMLSpy instance, and that it's visible
    StartXMLSpy_Click(null, null);

    // Open a file from the Examples folder installed with the product
    XMLSpy.Documents.OpenFile(strExamplesFolder + "OrgChart.pxf", false);
}

// Handler for the "Open ExpReport.xml" button
private void openExpReport_Click(object sender, EventArgs e)
{
    // Make sure there's a running XMLSpy instance, and that it's visible
    StartXMLSpy_Click(null, null);

    // Open a file from the Examples folder installed with the product
    XMLSpy.Documents.OpenFile(strExamplesFolder + "ExpReport.xml", false);
}

```

The file opened last will be the active file.

Iterating through Open Documents

The code snippet below (from the [AutomateXMLSpy example](#); see the file *Form1.cs*) shows how to iterate through open documents. A condition is then tested within the iteration loop, and the document view is switched between Text View and Authentic View.

```
// Handler for the "Toggle view mode" button
private void toggleView_Click(object sender, EventArgs e)
{
    // Make sure there's a running XMLSpy instance, and that it's visible
    StartXMLSpy_Click(null, null);

    // Iterate through open documents and toggle current view between text and
    authentic view.
    foreach (XMLSpyLib.Document doc in XMLSpy.Documents)
    {
        if (doc.CurrentViewMode == XMLSpyLib.SPYViewModes.spyViewText)
        {
            doc.SwitchViewMode(XMLSpyLib.SPYViewModes.spyViewAuthentic);
        }
        else
        {
            doc.SwitchViewMode(XMLSpyLib.SPYViewModes.spyViewText);
        }
    }
}
```

The [AutomateXMLSpy example](#) is located in the C# folder of the API Examples folder:

Windows XP	C:/Documents and Settings/<username>/My Documents/ Altova/XMLSpy20XX/Examples/API/
Windows Vista, Windows 7	C:/Users/<username>/Documents/ Altova/XMLSpy20XX/Examples/API/

You can compile and run the project from within Visual Studio 2008 or Visual Studio 2010.

Errors and COM Output Parameters

The code snippet below (from the [AutomateXMLSpy example](#)) shows how to handle errors and COM output parameters. The method [XMLSpy.ActiveDocument.IsValid\(ref strErrorText, ref nErrorNumber, ref errorData\)](#) uses output parameters that are used, in the code snippet below, to generate an error-message text.

The [AutomateXMLSpy example](#) (see the file *Form1.cs*) is located in the C# folder of the API Examples folder:

Windows XP	C:/Documents and Settings/<username>/My Documents/ Altova/XMLSpy20XX/Examples/API/
Windows Vista, Windows 7	C:/Users/<username>/Documents/ Altova/XMLSpy20XX/Examples/API/

You can compile and run the project from within Visual Studio 2008 or Visual Studio 2010.

Code snippet

```
// Handler for button "Validate"
private void validate_Click(object sender, EventArgs e)
{
    // COM errors are returned to C# as exceptions. We use a try/catch block to
```

```

handle them.
    try
    {
        // Method 'IsValid' is one of the few functions that uses output
parameters
        // Use 'object' type for these parameters
        object strErrorText = "";
        object nErrorNumber = 0;
        object errorData = null;

        if (!XMLSpy.ActiveDocument.IsValid(ref strErrorText, ref nErrorNumber,
ref errorData))
        {
            // The COM call succeeded but the document is not valid
            // A detailed description of the problem is returned in
strErrorText, nErrorNumber and errorData
            listBoxMessages.Items.Add("Document " + XMLSpy.ActiveDocument.Name
+ " is not valid.");
            listBoxMessages.Items.Add("\tErrorText : " + strErrorText);
            listBoxMessages.Items.Add("\tErrorNumber: " + nErrorNumber);
            listBoxMessages.Items.Add("\tElement      : " + (errorData != null ?
((XMLSpyLib.XMLData) errorData).TextValue : "null"));
        }
        else
        {
            // The COM call succeeded and the document is valid
            listBoxMessages.Items.Add("Document " + XMLSpy.ActiveDocument.Name
+ " is valid.");
        }
    }
    catch (Exception ex)
    {
        // The COM call was not successful
        // Probably no application instance has been started or no document is
open.
        listBoxMessages.Items.Add("Error validating active document: " +
ex.Message);
    }
}

```

Events

The code snippet below (from the [AutomateXMLSpy example](#)) lists the code for two event handlers. The [AutomateXMLSpy example](#) (see the file *Form1.cs*) is located in the C# folder of the API Examples folder:

Windows XP	C:/Documents and Settings/<username>/My Documents/ Altova/XMLSpy20XX/Examples/API/
Windows Vista, Windows 7	C:/Users/<username>/Documents/ Altova/XMLSpy20XX/Examples/API/

You can compile and run the project from within Visual Studio 2008 or Visual Studio 2010.

```

// Event handler for OnDocumentOpened event
private void handleOnDocumentOpened(XMLSpyLib.Document i_ipDocument)
{
    MessageBox.Show("Document " + i_ipDocument.Name + " was opened!");
}

// Remember if the event handler is currently registered.
private bool bEventHandlerIsRegistered = false;

// Handler for button 'OnDocuemntOpened Event On/Off'
private void toggleOnDocumentOpenedEvent_Click(object sender, EventArgs e)
{
    if (XMLSpy != null)

```

```

        {
            if ( bEventHandlerIsRegistered)
                XMLSpy.OnDocumentOpened -= new XMLSpyLib.
_IApplicationEvents_OnDocumentOpenedEventHandler( handleOnDocumentOpened) ;
            else
                XMLSpy.OnDocumentOpened += new XMLSpyLib.
_IApplicationEvents_OnDocumentOpenedEventHandler( handleOnDocumentOpened) ;

            bEventHandlerIsRegistered = !bEventHandlerIsRegistered;
        }
    }
}

```

Java

The Application API can be accessed from Java code. To allow accessing the XMLSpy automation server directly from Java code, the libraries listed below must reside in the classpath. They are installed in the folder: `JavaAPI` in the XMLSpy application folder.

- `AltovaAutomation.dll`: a JNI wrapper for Altova automation servers
- `AltovaAutomation.jar`: Java classes to access Altova automation servers
- `XMLSpyAPI.jar`: Java classes that wrap the XMLSpy automation interface
- `XMLSpyAPI_JavaDoc.zip`: a Javadoc file containing help documentation for the Java API

Note: In order to use the Java API, the DLL and Jar files must be on the Java Classpath.

Example Java project

An example Java project is supplied with your product installation. You can test the Java project and modify and use it as you like. For more details of the example Java project, see the section, [Example Java Project](#).

Rules for mapping the Application API names to Java

The rules for mapping between the Application API and the Java wrapper are as follows:

- **Classes and class names**
For every interface of the XMLSpy automation interface a Java class exists with the name of the interface.
- **Method names**
Method names on the Java interface are the same as used on the COM interfaces but start with a small letter to conform to Java naming conventions. To access COM properties, Java methods that prefix the property name with `get` and `set` can be used. If a property does not support write-access, no setter method is available. Example: For the `Name` property of the `Document` interface, the Java methods `getName` and `setName` are available.
- **Enumerations**
For every enumeration defined in the automation interface, a Java enumeration is defined with the same name and values.
- **Events and event handlers**
For every interface in the automation interface that supports events, a Java interface with the same name plus `'Event'` is available. To simplify the overloading of single events, a Java class with default implementations for all events is provided. The name

of this Java class is the name of the event interface plus 'DefaultHandler'. For example:

Application: Java class to access the application

ApplicationEvents: Events interface for the Application

ApplicationEventsDefaultHandler: Default handler for ApplicationEvents

Exceptions to mapping rules

There are some exceptions to the rules listed above. These are listed below:

Interface	Java name
Document, method SetEncoding	setFileEncoding
AuthenticView, method Goto	gotoElement
AuthenticRange, method Goto	gotoElement
AuthenticRange, method Clone	cloneRange

This section

This section explains how some basic XMLSpy functionality can be accessed from Java code. It is organized into the following sub-sections:

- [Example Java Project](#)
- [Application Startup and Shutdown](#)
- [Simple Document Access](#)
- [Iterations](#)
- [Use of Out-Parameters](#)
- [Event Handlers](#)

Example Java Project

The XMLSpy installation package contains an example Java project, located in the Java folder of the API Examples folder:

Windows XP	C:/Documents and Settings/<username>/My Documents/ Altova/XMLSpy20XX/Examples/API/
Windows Vista, Windows 7	C:/Users/<username>/Documents/ Altova/XMLSpy20XX/Examples/API/

This folder contains Java examples for the XMLSpy API. You can test it directly from the command line using the batch file `BuildAndRun.bat`, or you can compile and run the example project from within Eclipse. See below for instructions on how to use these procedures.

File list

The Java examples folder contains all the files required to run the example project. These files are listed below:

AltovaAutomation.dll	Java-COM bridge: DLL part
AltovaAutomation.jar	Java-COM bridge: Java library part

XMLSpyAPI.jar	Java classes of the XMLSpy API
RunXMLSpy.java	Java example source code
BuildAndRun.bat	Batch file to compile and run example code from the command line prompt. Expects folder where Java Virtual Machine resides as parameter.
.classpath	Eclipse project helper file
.project	Eclipse project file
XMLSpyAPI_JavaDoc.zip	Javadoc file containing help documentation for the Java API

What the example does

The example starts up XMLSpy and performs a few operations, including opening and closing documents. When done, XMLSpy stays open. You must close it manually.

- [Start XML Spy](#): Starts XMLSpy, which is registered as an automation server, or activates XMLSpy if it is already running.
- [Open OrgChart.pxf](#): Locates one of the example documents installed with XMLSpy and opens it.
- [Iteration and Changing the View Mode](#): Changes the view of all open documents to Text View. The code also shows how to iterate through open documents.
- [Iteration, validation, output parameters](#): Validates the active document and shows the result in a message box. The code shows how to use output parameters.
- [Event Handling](#): Shows how to handle XMLSpy events.
- [Shut down XMLSpy](#): Shuts down XMLSpy.

You can modify the example in any way you like and run it.

Running the example from the command line

To run the example from the command line, open a command prompt window, go to the Java folder of the API Examples folder (*see above for location*), and then type:

```
buildAndRun.bat "<Path-to-the-Java-bin-folder>"
```

The Java binary folder must be that of a JDK 1.5 or later installation on your computer.

Press the **Return** key. The Java source in `RunXMLSpy.java` will be compiled and then executed.

Loading the example in Eclipse

Open Eclipse and use the **Import | Existing Projects into Workspace** command to add the Eclipse project file (`.project`) located in the Java folder of the API Examples folder (*see above for location*). The project `RunXMLSpy` will then appear in your Package Explorer or Navigator.

Select the project and then the command **Run as | Java Application** to execute the example.

Note: You can select a class name or method of the Java API and press F1 to get help for that class or method.

Java source code listing

The Java source code in the example file `RunXMLSpy.java` is listed below with comments.

```

001 // Access general JAVA-COM bridge classes
002 import com.altova.automation.libs.*;
003
004 // Access XMLSpy Java-COM bridge
005 import com.altova.automation.XMLSpy.*;
006 import com.altova.automation.XMLSpy.Enums.SPYViewModes;
007
008 /**
009  * An example that starts XMLSpy COM server and performs view operations on it
010  * Feel free to extend
011  */
012 public class RunXMLSpy
013 {
014     public static void main(String[] args)
015     {
016         // An instance of the application.
017         Application xmlSpy = null;
018
019         // Instead of COM error handling, use Java exception mechanism
020         try
021         {
022             // Start XMLSpy as COM server
023             xmlSpy = new Application();
024
025             // COM servers start up invisible, so make it visible
026             xmlSpy.setVisible(true);
027
028             // Locate samples installed with the product
029             String strExamplesFolder =
030                 System.getenv("USERPROFILE") + "\\My
Documents\\Altova\\XMLSpy2012\\Examples\\";
031
032             // Open two example files
033             xmlSpy.getDocuments().openFile(strExamplesFolder + "OrgChart.pxf", false);
034             xmlSpy.getDocuments().openFile(strExamplesFolder + "ExpReport.xml", false);
035
036             // Iterate through open documents and set view mode to 'Text'.
037             for (Document doc: xmlSpy.getDocuments())
038                 if ( doc.getCurrentViewMode() != SPYViewModes.spyViewText)
039                     doc.switchViewMode(SPYViewModes.spyViewText);
040
041             // An alternative iteration mode is index-based
042             // COM indices are typically zero-based
043             Documents documents = xmlSpy.getDocuments();
044             for (int i = 1; i <= documents.getCount();
045                 i++)
046             {
047                 Document doc = documents.getItem(i);
048
049                 // Validation is one of the few methods to have output parameters.
050                 // The class JVariant is the correct type for parameters in these cases.
051                 // To get values back mark them with the by-reference flag.
052                 JVariant validationErrorText = new
053
054                 JVariant.JStringVariant("");
055
056                 validationErrorText.setByRefFlag();
057                 JVariant validationErrorCount = new
058
059                 JVariant.JIntVariant(0);
060
061                 validationErrorCount.setByRefFlag();
062                 JVariant validationErrorXMLData = new
063
064                 JVariant.JIDispatchVariant(0);
065

```

```

066         validationErrorXMLData.setByRefFlag();
067         if (!doc.isValid( validationErrorText, validationErrorCount,
validationErrorXMLData))
068             System.out.println("Document" + doc.getName() + " is not wellformed - " +
validationErrorText.getStringValue());
069         else
070             System.out.println("Document" + doc.getName() + " is wellformed.");
071     }
072
073     // The following lines attach to the document events using a default
implementation
074     // for the events and override one of its methods.
075     // If you want to override all document events it is better to derive your
listener class
076     // from DocumentEvents and implement all methods of this interface.
077     Document doc = xmlSpy.getActiveDocument();
078     doc.addListener( new
079 DocumentEventsDefaultHandler()
080     {
081         @Override
082         public boolean
083         onBeforeCloseDocument(Document i_ipDoc) throws AutomationException
084         {
085             System.out.println("Document
086 " + i_ipDoc.getName() + " requested closing.");
087
088             // Allow closing of document
089             return true;
090         }
091     });
092     doc.close(true);
093     doc = null;
094
095     System.out.println("Watch XMLSpy!");
096 }
097 catch ( AutomationException e)
098 {
099     // e.printStackTrace();
100 }
101 finally
102 {
103     // Make sure that XMLSpy can shut down properly.
104     if ( xmlSpy != null)
105         xmlSpy.dispose();
106
107     // Since the COM server was made visible and still is visible,
108     // it will keep running, and needs to be closed manually.
109     System.out.println("Now close XMLSpy!");
110 }
111 }
112 }
113 }
114 }
115 }

```

Application Startup and Shutdown

The code listings below show how the application can be started up and shut down.

Application startup

Before starting up the application, the appropriate classes must be imported (see below).

```

01 // Access general JAVA-COM bridge classes
02 import com.altova.automation.libs.*;
03
04 // Access XMLSpy Java-COM bridge
05 import com.altova.automation.XMLSpy.*;
06 import com.altova.automation.XMLSpy.Enums.SPYViewModes;

```

```
07
08 /**
09  * An example that starts XMLSpy COM server and performs view operations on it
10  * Feel free to extend
11  */
12 public class RunXMLSpy
13 {
14     public static void main(String[] args)
15     {
16         // An instance of the application.
17         Application xmlSpy = null;
18
19         // Instead of COM error handling, use Java exception mechanism
20         try
21         {
22             // Start XMLSpy as COM server
23             xmlSpy = new Application();
24             // COM servers start up invisible, so make it visible
25             xmlSpy.setVisible( true );
26
27             ...
28         }
29     }
30 }
```

Application shutdown

The application can be shut down as shown below.

```
01 {
02     // Allow shutdown of XMLSpy by releasing the UI.
03     xmlSpy.setVisible( true );
04
05     // Make sure that XMLSpy can shut down properly.
06     if ( xmlSpy != null )
07         xmlSpy.dispose();
08
09     // Since the COM server was made visible and still is visible,
10     // it will keep running, and needs to be closed manually.
11     System.out.println( "Now close XMLSpy!" );
12 }
```

Simple Document Access

The code listing below shows how to open a document.

```
1 // Locate samples installed with the product
2 String strExamplesFolder =
3     System.getenv( "USERPROFILE" ) + "\\My Documents\\Altova\\XMLSpy2012\\Examples\\";
4
5
6 // Open file
7 xmlSpy.getDocuments().openFile( strExamplesFolder + "OrgChart.pxf", false );
```

Iterations

The listing below shows how to iterate through open documents.

```

01 // Iterate through open documents and set view mode to 'Text'.
02 for (Document doc: xmlSpy.getDocuments())
03     if ( doc.getCurrentViewMode() != SPYViewModes.spyViewText)
04         doc.switchViewMode( SPYViewModes.spyViewText);
05
06 // An alternative iteration mode is index-based
07 // COM indices are typically zero-based
08 Documents documents = xmlSpy.getDocuments();
09 for (int i = 1; i <= documents.getCount();
10     i++)
11     {
12         Document doc = documents.getItem(i);
13         ...
14     }

```

Use of Out-Parameters

The code listing below iterates through open documents and validates each of them. For each validation, a message is generated using the output parameters of the Validation method.

```

01 // An alternative iteration mode is index-based
02 // COM indices are typically zero-based
03 Documents documents = xmlSpy.getDocuments();
04 for (int i = 1; i <= documents.getCount();
05     i++)
06     {
07         Document doc = documents.getItem(i);
08
09         // Validation is one of the few methods to have output parameters.
10         // The class JVariant is the correct type for parameters in these cases.
11         // To get values back mark them with the by-reference flag.
12         JVariant validationErrorText = new
13
14         JVariant.JStringVariant("");
15
16         validationErrorText.setByRefFlag();
17         JVariant validationErrorCount = new
18
19         JVariant.JIntVariant(0);
20
21         validationErrorCount.setByRefFlag();
22         JVariant validationErrorXMLData = new
23
24         JVariant.JIDispatchVariant(0);
25
26         validationErrorXMLData.setByRefFlag();
27         if (!doc.isValid( validationErrorText,
28
29             validationErrorCount, validationErrorXMLData))
30             System.out.println("Document
31
32             " + doc.getName() + " is not wellformed - " +
33
34             validationErrorText.getStringValue());
35         else
36             System.out.println("Document
37
38             " + doc.getName() + " is wellformed.");
39     }

```

Event Handlers

The listing below shows how to listen for and use events.

```

01 // The following lines attach to the document events using a default implementation
02 // for the events and override one of its methods.
03 // If you want to override all document events it is better to derive your listener
04 // from DocumentEvents and implement all methods of this interface.
05
06 Document doc = xmlSpy.getActiveDocument();
07 doc.addListener( new DocumentEventsDefaultHandler()
08 {
09     @Override
10     public boolean
11     onBeforeCloseDocument( Document i_ipDoc) throws AutomationException
12     {
13         System.out.println("Document " + i_ipDoc.getName() + " requested closing.");
14
15         // Allow closing of document
16         return true;
17     }
18 });
19 doc.close( true );
20 doc = null;

```

3.1.3 The DOM and XMLData

The `XMLData` interface gives you full access to the XML structure behind the current document with less methods than DOM and is much simpler. The `XMLData` interface is a minimalist approach to reading and modifying existing, or newly created XML data. You might however, want to use a DOM tree because you can access one from an external source or you just prefer the MSXML DOM implementation.

The `ProcessDOMNode()` and `ProcessXMLDataNode()` functions provided below convert any segments of an XML structure between `XMLData` and DOM.

To use the `ProcessDOMNode()` function:

- pass the root element of the DOM segment you want to convert in `objNode` and
- pass the plugin object with the `CreateChild()` method in `objCreator`

To use the `ProcessXMLDataNode()` function:

- pass the root element of the `XMLData` segment in `objXMLData` and
- pass the `DOMDocument` object created with MSXML in `xmlDoc`

```

////////////////////////////////////
// DOM To XMLData conversion
Function ProcessDOMNode( objNode, objCreator)
{
    var objRoot;
    objRoot = CreateXMLDataFromDOMNode( objNode, objCreator );

    If( objRoot ) {
        If( ( objNode.nodeValue != Null ) && ( objNode.nodeValue.length > 0 ) )
            objRoot.TextValue = objNode.nodeValue;
        // add attributes
        If( objNode.attributes ) {
            var Attribute;
            var oNodeList = objNode.attributes;

```

```

    For( var i = 0; i < oNodeList.length; i++) {
        Attribute = oNodeList.item(i);

        var newNode;
        newNode = ProcessDOMNode( Attribute, objCreator);

        objRoot.AppendChild( newNode);
    }
}
If( objNode.hasChildNodes) {
    try {
        // add children
        var Item;
        oNodeList = objNode.childNodes;

        For( var i = 0; i < oNodeList.length; i++) {
            Item = oNodeList.item(i);

            var newNode;
            newNode = ProcessDOMNode( Item, objCreator);

            objRoot.AppendChild( newNode);
        }
    } catch( err) {
    }
}
Return objRoot;
}

Function CreateXMLDataFromDOMNode( objNode, objCreator)
{
    var bSetName = True;
    var bSetValue = True;

    var nKind = 4;

    switch( objNode.nodeType) {
        Case 2: nKind = 5; break;
        Case 3: nKind = 6; bSetName = False; break;
        Case 4: nKind = 7; bSetName = False; break;
        Case 8: nKind = 8; bSetName = False; break;
        Case 7: nKind = 9; break;
    }
    var objNew = Null;
    objNew = objCreator.CreateChild( nKind);

    If( bSetName)
        objNew.Name = objNode.nodeName;

    If( bSetValue && ( objNode.nodeValue != Null))
        objNew.TextValue = objNode.nodeValue;

    Return objNew;
}
////////////////////////////////////
// XMLData To DOM conversion

Function ProcessXMLDataNode( objXMLData, xmlDoc)
{
    var objRoot;
    objRoot = CreateDOMNodeFromXMLData( objXMLData, xmlDoc);

    If( objRoot) {
        If( IsTextNodeEnabled( objRoot) && ( objXMLData.TextValue.length > 0))

```



```

objRoot.appendChild( xmlDoc.createTextNode( objXMLData.TextValue) );

If( objXMLData.HasChildren ) {
  try {
    var objChild;
    objChild = objXMLData.GetFirstChild(-1);

    While( True ) {
      If( objChild ) {
        var newNode;
        newNode = ProcessXMLDataNode( objChild, xmlDoc );

        If( newNode.nodeType == 2 ) {
          // child node is an attribute
          objRoot.attributes.setNamedItem( newNode );
        }
        Else
          objRoot.appendChild( newNode );
        objChild = objXMLData.GetNextChild();
      }
    }
    catch( err ) {
    }
  }
}
Return objRoot;
}

Function CreateDOMNodeFromXMLData( objXMLData, xmlDoc )
{
  switch( objXMLData.Kind ) {
    Case 4: Return xmlDoc.createElement( objXMLData.Name );
    Case 5: Return xmlDoc.createAttribute( objXMLData.Name );
    Case 6: Return xmlDoc.createTextNode( objXMLData.TextValue );
    Case 7: Return xmlDoc.createCDATASection( objXMLData.TextValue );
    Case 8: Return xmlDoc.createComment( objXMLData.TextValue );
    Case 9: Return
xmlDoc.createProcessingInstruction( objXMLData.Name, objXMLData.TextValue );
  }

  Return xmlDoc.createElement( objXMLData.Name );
}

Function IsTextNodeEnabled( objNode )
{
  switch( objNode.nodeType ) {
    Case 1:
    Case 2:
    Case 5:
    Case 6:
    Case 11: Return True;
  }
  Return False;
}

```

3.1.4 Obsolete: Authentic View Row operations

If the schema on which an XML document is based specifies that an element is repeatable, such a structure can be represented in Authentic View as a table. When represented as a table, rows and their contents can be manipulated individually, thereby allowing you to manipulate each of the repeatable elements individually. Such row operations would be performed by an external script.

If an external script is to perform row operations then two steps must occur:

- The first step checks whether the cursor is currently in a row using a property. Such a check could be, for example, `IsRowInsertEnabled`, which returns a value of either `TRUE` or `FALSE`.
- If the return value is `TRUE` then a row method, such as `RowAppend`, can be called. (`RowAppend` has no parameters and returns no value.)

The following is a list of properties and methods available for table operations. Each property returns a `BOOL`, and the methods have no parameter.

Property	Method	Table operations
<code>IsRowInsertEnabled</code>	<code>RowInsert</code> , superseded by <code>AuthenticRange.InsertRow</code>	Insert row operation
<code>IsRowAppendEnabled</code>	<code>RowAppend</code> , superseded by <code>AuthenticRange.AppendRow</code>	Append row operation
<code>IsRowDeleteEnabled</code>	<code>RowDelete</code> , superseded by <code>AuthenticRange.DeleteRow</code>	Delete row operation
<code>IsRowMoveUpEnabled</code>	<code>RowMoveUp</code> , superseded by <code>AuthenticRange.MoveRowUp</code>	Move XML data up one row
<code>IsRowMoveDownEnabled</code>	<code>RowMoveDown</code> , superseded by <code>AuthenticRange.MoveRowDown</code>	Move XML data down one row
<code>IsRowDuplicateEnabled</code>	<code>RowDuplicate</code> , superseded by <code>AuthenticRange.DuplicateRow</code>	Duplicate currently selected row

3.2 Interfaces

Object Hierarchy

[Application](#)

[SpyProject](#)

[SpyProjectItems](#)

[SpyProjectItem](#)

[Documents](#)

[Document](#)

[GridView](#)

[AuthenticView](#)

[AuthenticRange](#)

[AuthenticDataTransfer](#) (previously DocEditDataTransfer)

[OldAuthenticView](#) (previously DocEditView, **now obsolete**, superseded by

[AuthenticView](#) and [AuthenticRange](#))

[AuthenticSelection](#) (previously DocEditSelection, **now obsolete**,
superseded by [AuthenticRange](#))

[AuthenticEvent](#) (previously DocEditEvent, **now obsolete**)

[AuthenticDataTransfer](#) (previously DocEditDataTransfer)

[TextView](#)

[XMLData](#)

[Dialogs](#)

[CodeGeneratorDlg](#)

[FileSelectionDlg](#)

[SchemaDocumentationDlg](#)

[GenerateSampleXMLDlg](#)

[DTDSchemaGeneratorDlg](#)

[FindInFilesDlg](#)

[WSDLDocumentationDlg](#)

[WSDL20DocumentationDlg](#)

[XBRLDocumentationDlg](#)

[DatabaseConnection](#)

[ExportSettings](#)

[TextImportExportSettings](#)

[ElementList](#)

[ElementListItem](#)

[Enumerations](#)

Description

This chapter contains the reference of the XMLSpy 1.5 Type Library.

Most of the given examples are written in VisualBasic. These code snippets assume that there is a variable defined and set, called **objSpy of type Application**. There are also some code samples written in JavaScript.

3.2.1 Application

See also

Methods

[GetDatabaseImportElementList](#)

[GetDatabaseSettings](#)

[GetDatabaseTables](#)

[ImportFromDatabase](#)

[CreateXMLSchemaFromDBStructure](#)

[GetTextImportElementList](#)
[GetTextImportExportSettings](#)
[ImportFromText](#)

[ImportFromWord](#)

[ImportFromSchema](#)

[GetExportSettings](#)

[NewProject](#)

[OpenProject](#)

[AddMacroMenuItem](#)

[ClearMacroMenu](#)

[ShowForm](#)

[ShowApplication](#)

[URLDelete](#)

[URLMakeDirectory](#)

[FindFiles](#)

[Quit](#)

Properties

[Application](#)

[Parent](#)

[ActiveDocument](#)

[Documents](#)

[CurrentProject](#)

[Dialogs](#)

[WarningNumber](#)

[WarningText](#)

[Status](#)

[MajorVersion](#)

[MinorVersion](#)

[Edition](#)

[IsAPISupported](#)

[ServicePackVersion](#)

Description

Application is the root for all other objects. It is the only object you can create by `CreateObject` (VisualBasic) or other similar COM related functions.

Example

```
Dim objSpy As Application
Set objSpy = CreateObject("XMLSpy.Application")
```

Events

OnBeforeOpenDocument

See also

Event: *OnBeforeOpenDocument*(*objDialog* as [FileSelectionDlg](#))

Description

This event gets fired whenever a document gets opened via the OpenFile or OpenURL menu command. It is sent after a document file has been selected but before the document gets opened. The file selection dialog object is initialized with the name of the selected document file. You can modify this selection. To continue the opening of the document leave the [FileSelectionDlg.Action](#) property of *io_objDialog* at its default value [spyDAbcOK](#). To abort the opening of the document set this property to [spyDAbcCancel](#).

Examples

Given below are examples of how this event can be scripted.

XMLSpy scripting environment - VBScript:

```
Function On_BeforeOpenDocument(objDialog)
EndFunction
```

XMLSpy scripting environment - JScript:

```
function On_BeforeOpenDocument(objDialog)
{
}
```

XMLSpy IDE Plugin:

```
IXMLSpyPlugin.OnEvent (26, ...) //nEventId=26
```

OnBeforeOpenProject

See also

Event: *OnBeforeOpenProject*(*objDialog* as [FileSelectionDlg](#))

Description

This event gets fired after a project file has been selected but before the project gets opened. The file selection dialog object is initialized with the name of the selected project file. You can modify this selection. To continue the opening of the project leave the [FileSelectionDlg.Action](#) property of *io_objDialog* at its default value [spyDAbcOK](#). To abort the opening of the project set this property to [spyDAbcCancel](#).

Examples

Given below are examples of how this event can be scripted.

XMLSpy scripting environment - VBScript:

```
Function On_BeforeOpenProject(objDialog)
EndFunction
```

XMLSpy scripting environment - JScript:

```
function On_BeforeOpenProject(objDialog)
```

```
{
}
```

XMLSpy IDE Plugin:

```
IXMLSpyPlugIn.OnEvent (25, ...) //nEventId=25
```

OnDocumentOpened**See also**

Event: OnDocumentOpened(*objDocument* as [Document](#))

Description

This event gets fired whenever a document opens in XMLSpy. This can happen due to opening a file with the OpenFile or OpenURL dialog, creating a new file or dropping a file onto XMLSpy. The new document gets passed as parameter. The operation cannot be canceled.

Examples

Given below are examples of how this event can be scripted.

XMLSpy scripting environment - VBScript:

```
Function On_OpenDocument(objDocument)
EndFunction
```

XMLSpy scripting environment - JScript:

```
function On_OpenDocument(objDocument)
{
}
```

XMLSpy IDE Plugin:

```
IXMLSpyPlugIn.OnEvent (7, ...) //nEventId=7
```

OnProjectOpened**See also**

Event: OnProjectOpened(*objProject* as [SpyProject](#))

Description

This event gets fired whenever a project gets opened in XMLSpy. The new project gets passed as parameter.

Examples

Given below are examples of how this event can be scripted.

XMLSpy scripting environment - VBScript:

```
Function On_OpenProject(objProject)
EndFunction
```

XMLSpy scripting environment - JScript:

```
function On_OpenProject(objProject)
{
}
```

```
}
```

XMLSpy IDE Plugin:

```
IXMLSpyPlugin.OnEvent (6, ...) //nEventId=6
```

ActiveDocument**See also**

Property: [ActiveDocument](#) as [Document](#)

Description

Reference to the active document. If no document is open, `ActiveDocument` is null (nothing).

Errors

- 1111 The application object is no longer valid.
- 1100 Invalid address for the return parameter was specified.

AddMacroMenuItem**See also**

Method: `AddMacroMenuItem`(*strMacro* as String,*strDisplayText* as String)

Description

Adds a menu item to the **Tools** menu. This new menu item invokes the macro defined by `strMacro`. See also "[Calling macros](#)" from XMLSpy".

Errors

- 1111 The application object is no longer valid.
- 1100 Invalid parameter or invalid address for the return parameter was specified.
- 1108 Number of macro items is limited to 16 items.

Application**See also**

Property: [Application](#) as [Application](#) (read-only)

Description

Accesses the XMLSpy application object.

Errors

- 1111 The application object is no longer valid.
- 1100 Invalid address for the return parameter was specified.

ClearMacroMenu**See also**

Method: `ClearMacroMenu`()

Return Value

None

Description

Removes all menu items from the **Tools** menu. See also Calling macros from XMLSpy".

Errors

1111 The application object is no longer valid.

CreateXMLSchemaFromDBStructure**See also**

Method: `CreateXMLSchemaFromDBStructure`(`pImportSettings` as `DatabaseConnection`, `pTables` as `ElementList`)

Description

`CreateXMLSchemaFromDBStructure` creates from a database specified in `pImportSettings` for the defined tables in `pTables` new XML Schema document(s) describing the database tables structure.

The parameter `pTables` specifies which table structures the XML Schema document should contain. This parameter can be NULL, specifying that all table structures will be exported.

See also [GetDataBaseTables](#).

Errors

1112 Invalid database specified.

1120 Database import failed.

CurrentProject**See also**

Property: `CurrentProject` as [SpyProject](#)

Description

Reference to the active document. If no project is open, `CurrentProject` is null (nothing).

Errors

1111 The application object is no longer valid.

1100 Invalid address for the return parameter was specified.

Dialogs**See also**

Property: `Dialogs` as [Dialogs](#) (read-only)

Description

Access the built-in dialogs of XMLSpy.

Errors

1111 The application object is no longer valid.

1100 Invalid address for the return parameter was specified.

Documents

See also

Property: [Documents](#) as [Documents](#)

Description

Collection of all open documents. See also Simple document access.

Errors

- 1111 The application object is no longer valid.
- 1100 Invalid address for the return parameter was specified.

Edition

See also

Property: [Edition](#) as String

Description

Returns the edition of the application. Eg: Enterprise, Professional, Basic

Errors

- 1111 The application object is no longer valid.
- 1100 Invalid address for the return parameter was specified.

FindInFiles

See also

Method: [FindInFiles](#)([pSettings](#) as [FindInFilesDlg](#)) as [FindInFilesResults](#)

Description

Returns a [FindInFilesResults](#) object containing information about the files that matched the specified settings.

Errors

- 1111 The application object is no longer valid.
- 1100 Invalid address for the return parameter was specified.

GetDatabaseImportElementList

See also

Method: [GetDatabaseImportElementList](#)([pImportSettings](#) as [DatabaseConnection](#)) as [ElementList](#)

Description

The function returns a collection of [ElementListItems](#) where the properties [ElementListItem.Name](#) contain the names of the fields that can be selected for import and the properties [ElementListItem.ElementKind](#) are initialized either to [spyXMLDataAttr](#) or [spyXMLDataElement](#), depending on the value passed in [DatabaseConnection.AsAttributes](#). This list serves as a filter to what finally gets imported by a future call to [ImportFromDatabase](#). Use [ElementList.RemoveElement](#) to exclude fields from import.

Properties mandatory to be filled out for the database connection are one of [DatabaseConnection.File](#), [DatabaseConnection.ADOConnection](#) and [DatabaseConnection.ODBCConnection](#), as well as [DatabaseConnection.SQLSelect](#). Use the property [DatabaseConnection.AsAttributes](#) to initialize [ElementListItem.ElementKind](#) of the resulting element list to either *spyXMLDataAttr* or *spyXMLDataElement*, respectively.

Example

See example at [ImportFromDatabase](#).

Errors

- 1111 The application object is no longer valid.
- 1100 Invalid parameter or invalid address for the return parameter was specified.
- 1107 Import from database failed.
- 1112 Invalid database specified.
- 1114 Select statement is missing.
- 1119 database element list import failed.

GetDatabaseSettings

See also

Method: [GetDatabaseSettings\(\)](#) as [DatabaseConnection](#)

Description

[GetDatabaseSettings](#) creates a new object of database settings. The object is used to specify database connection parameters for the methods [GetDatabaseTables](#), [GetDatabaseImportElementList](#), [ImportFromDatabase](#), [ImportFromSchema](#) and [ExportToDatabase](#).

Example

See example of [ImportFromDatabase](#).

Errors

- 1111 The application object is no longer valid.
- 1100 Invalid address for the return parameter was specified.

GetDatabaseTables

See also

Method: [GetDatabaseTables\(*pImportSettings* as \[DatabaseConnection\]\(#\)\)](#) as [ElementList](#)

Description

[GetDatabaseTables](#) reads the table names from the database specified in *pImportSettings*. Properties mandatory to be filled out for the database connection are one of [DatabaseConnection.File](#), [DatabaseConnection.ADOConnection](#) and [DatabaseConnection.ODBCConnection](#). All other properties are ignored. The function returns a collection of [ElementListItems](#) where the properties [ElementListItem.Name](#) contain the names of tables stored in the specified database. The remaining properties of [ElementListItem](#) are unused.

Errors

- 1111 The application object is no longer valid.
- 1100 Invalid parameter or invalid address for the return parameter was specified.
- 1112 Invalid database specified.
- 1113 Error while reading database table information.
- 1118 Database table query failed.

Example

```
Dim objImpSettings As DatabaseConnection
Set objImpSettings = objSpy.GetDatabaseSettings
objImpSettings.ADOConnection = TxtADO.Text

'store table names in list box
ListTables.Clear

Dim objList As ElementList
Dim objItem As ElementListItem
On Error GoTo ErrorHandler
Set objList = objSpy.GetDatabaseTables(objImpSettings)

For Each objItem In objList
    ListTables.AddItem objItem.Name
Next
```

GetExportSettings

See also

Method: [GetExportSettings\(\)](#) as [ExportSettings](#) (read-only)

Description

[GetExportSettings](#) creates a new object of common export settings. This object is used to pass the parameters to the export functions and defines the behaviour of the export calls. See also the export functions from [Document](#) and the examples at [Import and Export](#).

Errors

- 1111 The application object is no longer valid.
- 1100 Invalid address for the return parameter was specified.

GetTextImportElementList

See also

Method: [GetTextImportElementList](#)([pImportSettings](#) as [TextImportExportSettings](#)) as [ElementList](#)

Description

[GetTextImportElementList](#) retrieves importing information about the text-file as specified in [pImportSettings](#). The function returns a collection of [ElementListItems](#) where the properties [ElementListItem.Name](#) contain the names of the fields found in the file. The values of remaining properties are undefined.

If the text-file does not contain a column header, set [pImportSettings.HeaderRow](#) to [false](#). The resulting element list will contain general column names like 'Field1' and so on.

See also [Import and export of data](#).

Errors

- 1111 The application object is no longer valid.
- 1100 Invalid parameter or invalid address for the return parameter was specified.
- 1107 Import from database failed.
- 1115 Error during text element list import. Cannot create parser for import file.
- 1116 Error during text element list import.

Example

```

' -----
' VBA client code fragment - import selected fields from text file
' -----
Dim objImpSettings As TextImportExportSettings
Set objImpSettings = objSpy.GetTextImportExportSettings

objImpSettings.ImportFile = "C:\ImportMe.txt"
objImpSettings.HeaderRow = False

Dim objList As ElementList
Set objList = objSpy.GetTextImportElementList(objImpSettings)

'exclude first column
objList.RemoveItem 1

Dim objImpDoc As Document
On Error Resume Next
Set objImpDoc = objSpy.ImportFromText(objImpSettings, objList)
CheckForError

```

GetTextImportExportSettings**See also**

Method: [GetTextImportExportSettings\(\)](#) as [TextImportExportSettings](#) (read-only)

Description

[GetTextImportExportSettings](#) creates a new object of common import and export settings for text files. See also the example for [Application.GetTextImportElementList](#) and [Import and Export](#).

See also [Import and export of data](#).

Errors

- 1111 The application object is no longer valid.
- 1100 Invalid address for the return parameter was specified.

ImportFromDatabase**See also**

Method: [ImportFromDatabase\(pImportSettings as DatabaseConnection
ElementList as ElementList\)](#) as [Document](#)

Return Value

Creates a new document containing the data imported from the database.

Description

[ImportFromDatabase](#) imports data from a database as specified in [pImportSettings](#) and creates a new document containing the data imported from the database. Properties mandatory to be

filled out are one of [DatabaseConnection.File](#), [DatabaseConnection.ADOConnection](#) or [DatabaseConnection.ODBCConnection](#) and [DatabaseConnection.SQLSelect](#). Additionally, you can use [DatabaseConnection.AsAttributes](#), [DatabaseConnection.ExcludeKeys](#), [DatabaseConnection.IncludeEmptyElements](#) and [NumberDateTimeFormat](#) to further parameterize import.

The parameter `pElementList` specifies which fields of the selected data gets written into the newly created document, and which are created as elements and which as attributes. This parameter can be NULL, specifying that all selected fields will be imported as XML elements.

See [GetDatabaseSettings](#) and [GetDatabaseImportElementList](#) for necessary steps preceding any import of data from a database.

Errors

- 1111 The application object is no longer valid.
- 1100 Invalid parameter or invalid address for the return parameter was specified.
- 1107 Import from database failed.
- 1112 Invalid database specified.
- 1114 Select statement is missing.
- 1117 Transformation to XML failed.
- 1120 Database import failed.

Example

```
Dim objImpSettings As DatabaseConnection
Set objImpSettings = objSpy.GetDatabaseSettings

objImpSettings.ADOConnection = strADOConnection
objImpSettings.SQLSelect = "SELECT * FROM MyTable"

Dim objDoc As Document
On Error Resume Next
Set objDoc = objSpy.ImportFromDatabase(objImpSettings,
objSpy.GetDatabaseImportElementList(objImpSettings))
' CheckForError here
```

ImportFromSchema

See also

Method: `ImportFromSchema(pImportSettings as DatabaseConnection, strTable as String, pSchemaDoc as Document) as Document`

Return Value

Creates a new document filled with data from the specified database as specified by the schema definition in *pSchemaDoc*.

Description

`ImportFromSchema` imports data from a database specified in `pImportSettings`. Properties mandatory to be filled out are one of [DatabaseConnection.File](#), [DatabaseConnection.ADOConnection](#) or [DatabaseConnection.ODBCConnection](#). Additionally, you can use [DatabaseConnection.AsAttributes](#), [DatabaseConnection.ExcludeKeys](#) and [NumberDateTimeFormat](#) to further parameterize import. All other properties get ignored.

`ImportFromSchema` does not use and explicit SQL statement to select the data. Instead, it

expects a structure definition of the document to create in form of an XML schema document in *pSchemaDoc*. From this definition the database select statement is automatically deduced. Specify in *strTable* the table name of the import root that will become the root node in the new document.

See [GetDatabaseSettings](#) and [GetDatabaseTables](#) for necessary steps preceding an import from a database based on a schema definition. To create the schema definition file use command 'create database schema' from the 'convert' menu of XMLSpy.

Errors

- 1111 The application object is no longer valid.
- 1100 Invalid parameter or invalid address for the return parameter was specified.
- 1107 Import from database failed.
- 1112 Invalid database specified.
- 1120 Database import failed.
- 1121 Could not create validator for the specified schema.
- 1122 Failed parsing schema for database import.

ImportFromText

See also

Method: `ImportFromText(pImportSettings as TextImportExportSettings, pElementList as ElementList) as Document`

Description

`ImportFromText` imports the text file as specified in `pImportSettings`. The parameter `pElementList` can be used as import filter. Either pass the list returned by a previous call to [GetTextImportElementList](#) or `nil` to import all columns. To avoid import of unnecessary columns use [ElementList.RemoveElement](#) to remove the corresponding field names from `pElementList` before calling `ImportFromText`.

The method returns the newly created document containing the imported data. This document is the same as the active document of XMLSpy.

See also [Import and export of data](#).

Errors

- 1111 The application object is no longer valid.
- 1100 Invalid parameter or invalid address for the return parameter was specified.
- 1107 Import from text file failed.
- 1117 Transformation to XML failed.

Example

```
' -----
' VBA client code fragment - import from text file
' -----

Dim objImpSettings As TextImportExportSettings
Set objImpSettings = objSpy.GetTextImportExportSettings

objImpSettings.ImportFile = strFileName
objImpSettings.HeaderRow = False

Dim objImpDoc As Document
On Error Resume Next
Set objImpDoc = objSpy.ImportFromText(objImpSettings,
    objSpy.GetTextImportElementList(objImpSettings))
```

CheckForError

ImportFromWord

See also

Method: `ImportFromWord(strFile as String)` as [Document](#)

Description

`ImportFromWord` imports the MS-Word Document `strFile` into a new XML document.

Errors

- 1111 The application object is no longer valid.
- 1100 Invalid parameter or invalid address for the return parameter was specified.
Import from document failed.

IsAPISupported

See also

Property: `IsAPISupported` as Boolean

Description

Returns whether the API is supported in this version or not.

Errors

- 1111 The application object is no longer valid.
- 1100 Invalid address for the return parameter was specified.

MajorVersion

See also

Property: `MajorVersion` as Integer

Description

Returns the application version's major number.

Errors

- 1111 The application object is no longer valid.
- 1100 Invalid address for the return parameter was specified.

MinorVersion

See also

Property: `MinorVersion` as Integer

Description

Returns the application version's minor number.

Errors

- 1111 The application object is no longer valid.
- 1100 Invalid address for the return parameter was specified.

NewProject

See also

Method: `NewProject`(*strPath* as String, *bDiscardCurrent* as Boolean)

Description

`NewProject` creates a new project.

If there is already a project open that has been modified and `bDiscardCurrent` is false, then `NewProject` fails.

Errors

- 1111 The application object is no longer valid.
- 1102 A project is already open but *bDiscardCurrent* is *true*.
- 1103 Creation of new project failed.

OpenProject

See also

Method: `OpenProject`(*strPath* as String, *bDiscardCurrent* as Boolean, *bDialog* as Boolean)

Parameters

strPath

Path and file name of the project to open. Can be empty if `bDialog` is true.

bDiscardCurrent

Discard currently open project and possible lose changes.

bDialog

Show dialogs for user input.

Return Value

None

Description

`OpenProject` opens an existing project. If there is already a project open that has been modified and `bDiscardCurrent` is false, then `OpenProject` fails.

Errors

- 1111 The application object is no longer valid.
- 1100 Invalid parameter or invalid address for the return parameter was specified.
- 1101 Cannot open specified project.
- 1102 A project is already open but *bDiscardCurrent* is *true*.

Parent

See also

Property: `Parent` as [Application](#) (read-only)

Description

Accesses the XMLSpy application object.

Errors

- 1111 The application object is no longer valid.
- 1100 Invalid address for the return parameter was specified.

Quit**See also**

Method: [Quit\(\)](#)

Return Value

None

Description

This method terminates XMLSpy. All modified documents will be closed without saving the changes. This is also true for an open project.

If XMLSpy was automatically started as an automation server by a client program, the application will not shut down automatically when your client program shuts down if a project or any document is still open. Use the Quit method to ensure automatic shut-down.

Errors

- 1111 The application object is no longer valid.

ReloadSettings**See also**

Method: [ReloadSettings](#)

Return Value**Description**

The application settings are reloaded from the registry.

Available with TypeLibrary version 1.5

Errors

- 1111 The application object is no longer valid.

RunMacro**See also**

Method: [RunMacro](#)(*strMacro* as String)

Return Value**Description**

Calls the specified macro either from the project scripts (if present) or from the global scripts.

Available with TypeLibrary version 1.5

Errors

1111 The application object is no longer valid.

ScriptingEnvironment**See also**

Property: [ScriptingEnvironment](#) as IUnknown (read-only)

Description

Reference to any active scripting environment. This property makes it possible to access the TypeLibrary of the XMLSpyFormEditor.exe application which is used as the current scripting environment.

Available with TypeLibrary version 1.5

Errors

1111 The application object is no longer valid.

1100 Invalid address for the return parameter was specified.

ServicePackVersion**See also**

Property: [ServicePackVersion](#) as Long

Description

Returns the Service Pack version number of the application. Eg: 1 for 2010 R2 SP1

Errors

1111 The application object is no longer valid.

1100 Invalid address for the return parameter was specified.

ShowApplication**See also**

Method: [ShowApplication](#)([bShow](#) as Boolean)

Return Value

None

Description

The method shows ([bShow = True](#)) or hides ([bShow = False](#)) XMLSpy.

Errors

1110 The application object is no longer valid.

ShowFindInFiles**See also**

Method: [ShowFindInFiles](#)([pSettings](#) as [FindFilesDlg](#)) as Boolean

Return Value

Returns false if the user pressed the Cancel button, true otherwise.

Description

Displays the FindInFiles dialog preset with the given settings. The user modifications of the settings are stored in the passed dialog object.

Errors

- 1111 The application object is no longer valid.
- 1100 Invalid parameter or invalid address for the return parameter was specified.

ShowForm**See also**

Method: `ShowForm(strFormName as String) as Long`

Return Value

Returns zero if the user pressed a Cancel button or the form calls `TheView.Cancel()` .

Description

Displays the form `stFormName` .

Forms, event handlers and macros can be created with the Scripting Environment. Select "Switch to scripting environment" from the **Tools** menu to invoke the Scripting Environment.

Errors

- 1111 The application object is no longer valid.
- 1100 Invalid parameter or invalid address for the return parameter was specified.

Status**See also**

Property: `Status` as [ENUMApplicationStatus](#)

Description

Returns the current status of the running application.

Errors

- 1111 The application object is no longer valid.
- 1100 Invalid address for the return parameter was specified.

URLDelete**See also**

Method: `URLDelete(strURL as String, strUser as String, strPassword as String)`

Return Value

None

Description

The method deletes the file at the URL `stURL` .

Errors

- 1111 The application object is no longer valid.
- 1109 Error deleting file at specified URL.

URLMakeDirectory

See also

Method: `URLMakeDirectory( strURL as String, strUser as String, strPassword as String )`

Return Value

None

Description

The method creates a new directory at the URL `strURL` .

Errors

- 1111 The application object is no longer valid.
- 1100 Invalid parameter specified.

Visible

See also

Property: `Visible` as `VARIANT_BOOL`

Description

Sets or gets the visibility attribute of XMLSpy. This standard automation property makes usage of [ShowApplication](#) obsolete.

Errors

- 1110 The application object is no longer valid.
- 1100 Invalid address for the return parameter was specified.

WarningNumber

See also

Property: `WarningNumber` as integer

Description

Some methods fill the property `WarningNumber` with additional information if an error occurs.

Currently just [Documents.OpenFile](#) fills this property.

Errors

- 1111 The application object is no longer valid.
- 1100 Invalid address for the return parameter was specified.

WarningText

See also

Property: `WarningText` as String

Description

Some methods fill the property `WarningText` with additional information if an error occurs.

Currently just [Documents.OpenFile](#) fills this property.

Errors

- 1111 The application object is no longer valid.
- 1100 Invalid address for the return parameter was specified.

3.2.2 AuthenticContextMenu

The context menu interface provides the mean for the user to customize the context menus shown in Authentic. The interface has the methods listed in this section.

CountItems

Method: `CountItems()` `ntItems` as long

Return Value

Returns the number of menu items.

Errors

- 2501 Invalid object.

DeleteItem

Method: `DeleteItem(IndexPosition as long)`

Return Value

Deletes the menu item that has the index position submitted in the first parameter.

Errors

- 2501 Invalid object
- 2502 Invalid index

GetItemText

Method: `GetItemText(IndexPosition as long)` `MenuItemName` as string

Return Value

Gets the name of the menu item located at the index position submitted in the first parameter.

Errors

- 2501 Invalid object
- 2502 Invalid index

InsertItem

Method: `InsertItem(IndexPosition as long, MenuItemName as string, MacroName as string)`

Return Value

Inserts a user-defined menu item at the position in the menu specified in the first parameter and having the name submitted in the second parameter. The menu item will start a macro, so a valid macro name

must be submitted.

Errors

- 2501 Invalid object
- 2502 Invalid index
- 2503 No such macro
- 2504 Internal error

SetItemText

Method: `SetItemText`(IndexPosition as long, MenuItemName as string)

Return Value

Sets the name of the menu item located at the index position submitted in the first parameter.

Errors

- 2501 Invalid object
- 2502 Invalid index

3.2.3 AuthenticDataTransfer

Renamed from **DocEditDataTransfer** to **AuthenticDataTransfer**

The `DocEditView` object is renamed to `ObAuthenticView`.
`DocEditSelection` is renamed to `AuthenticSelection`.
`DocEditEvent` is renamed to `AuthenticEvent`.
`DocEditDataTransfer` is renamed to `AuthenticDataTransfer`.

Their usage—except for `AuthenticDataTransfer`—is no longer recommended. We will continue to support existing functionality for a yet undefined period of time but no new features will be added to these interface. All functionality available up to now in `DocEditView`, `DocEditSelection`, `DocEditEvent` and `DocEditDataTransfer` is now available via `AuthenticView`, `AuthenticRange` and `AuthenticDataTransfer`. Many new features have been added.

For examples on migrating from DocEdit to Authentic see the description of the different methods and properties of the different DocEdit objects.

See also

Methods

[getData](#)

Properties

[dropEffect](#)
[ownDrag](#)
[type](#)

Description

The events `OnDragOver` and `OnBeforeDrop` provide information about the object being dragged

with an instance of type `AuthenticDataTransfer`. It contains a description of the dragged object and its content. The latter is available either as string or a pointer to a COM object supporting the `IUnknown` interface.

dropEffect

See also

Property: `dropEffect` as long

Description

The property stores the drop effect from the default event handler. You can set the drop effect if you change this value and return `TRUE` for the event handler (or set [AuthenticEvent.cancelBubble](#) to `TRUE` if you are still using the now obsolete `AuthenticEvent` interface).

Errors

2101 Invalid address for the return parameter was specified.

getData

See also

Method: `getData()` as Variant

Description

Retrieve the data associated with the dragged object. Depending on [AuthenticDataTransfer.type](#), that data is either a string or a COM interface pointer of type `IUnknown`.

Errors

2101 Invalid address for the return parameter was specified.

ownDrag

See also

Property: `ownDrag` as Boolean (read-only)

Description

The property is `TRUE` if the current dragging source comes from inside Authentic View.

Errors

2101 Invalid address for the return parameter was specified.

type

See also

Property: `type` as String (read-only)

Description

Holds the type of data you get with the [DocEditDataTransfer.getData](#) method.

Currently supported data types are:

OWN	data from Authentic View itself
TEXT	plain text
UNICODETEXT	plain text as UNICODE

Errors

2101 Invalid address for the return parameter was specified.

3.2.4 AuthenticEventContext

The `EventContext` interface gives access to many properties of the context in which a macro is executed.

EvaluateXPath

Method: `EvaluateXPath (strExpression as string)` as strValue as string

Return Value

The method evaluates the XPath expression in the context of the node within which the event was triggered and returns a string.

Description

`EvaluateXPath()` executes an XPath expressions with the given event context. The result is returned as string, in the case of a sequence it is a space-separated string.

Errors

2201 Invalid object.
 2202 No context.
 2209 Invalid parameter.
 2210 Internal error.
 2211 XPath error.

GetEventContextType

Method: `GetEventContextType ()` Type as `AuthenticEventContextType` enumeration

Return Value

Returns the context node type.

Description

`GetEventContextType` allows the user to determine whether the macro is in an XML node or in an XPath atomic item context. The enumeration `AuthenticEventContextType` is defined as follows:

```
authenticEventContextXML,
authenticEventContextAtomicItem,
authenticEventContextOther
```

If the context is a normal XML node, the `GetXMLNode()` function gives access to it (returns `NULL` if not).

Errors

2201 Invalid object.
 2202 No context.
 2209 Invalid parameter.

GetNormalizedTextValue

Method: `GetNormalizedTextValue()` strValue as string

Return Value

Returns the value of the current node as string

Errors

- 2201 Invalid object.
- 2202 No context.
- 2203 Invalid context
- 2209 Invalid parameter.

GetVariableValue

Method: `GetVariableValue(strName as string)` strValue as string

Return Value

Gets the value of the variable submitted as the parameter.

Description

`GetVariableValue` gets the variable's value in the scope of the context.

```
nZoom = parseInt( AuthenticView.EventContext.GetVariableValue( 'Zoom' ) );
if ( nZoom > 1 )
{
    AuthenticView.EventContext.SetVariableValue( 'Zoom', nZoom - 1 );
}
```

Errors

- 2201 Invalid object.
- 2202 No context.
- 2204 No such variable in scope
- 2205 Variable cannot be evaluated
- 2206 Variable returns sequence
- 2209 Invalid parameter

GetXMLNode

Method: `GetXMLNode()` Node as XMLData object

Return Value

Returns the context XML node or `NULL`

Errors

- 2201 Invalid object.
- 2202 No context.
- 2203 Invalid context
- 2209 Invalid parameter.

IsAvailable

Method: `IsAvailable()` as Boolean

Return Value

Returns true if `EventContext` is set, false otherwise.

Errors

2201 Invalid object.

SetVariableValue

Method: `SetVariableValue(strName as string, strValue as string)`

Return Value

Sets the value (second parameter) of the variable submitted in the first parameter.

Description

`SetVariableValue` sets the variable's value in the scope of the context.

```
nZoom = parseInt( AuthenticView.EventContext.GetVariableValue( 'Zoom' ) );
if ( nZoom > 1 )
{
    AuthenticView.EventContext.SetVariableValue( 'Zoom', nZoom - 1 );
}
```

Errors

2201 Invalid object.
 2202 No context.
 2204 No such variable in scope
 2205 Variable cannot be evaluated
 2206 Variable returns sequence
 2207 Variable read-only
 2208 No modification allowed

3.2.5 AuthenticRange

See also

The first table lists the properties and methods of `AuthenticRange` that can be used to navigate through the document and select specific portions.

Properties

[Apptatin](#)
[FirstTextPosition](#)
[FirstXMLData](#)
[FirstXMLDataOffset](#)
[LastTextPosition](#)
[LastXMLData](#)
[LastXMLDataOffset](#)
[Parent](#)

[C.bne](#)
[CollapseToBegin](#)
[CollapseToEnd](#)
[ExpandTo](#)
[Goto](#)
[GotoNext](#)
[GotoPrevious](#)
[IsEmpty](#)
[IsEqual](#)

Methods

[MoveBegin](#)
[MoveEnd](#)
[NextCursorPosition](#)
[PreviousCursorPosition](#)
[Select](#)
[SelectNext](#)
[SelectPrevious](#)
[SetFromRange](#)

The following table lists the content modification methods, most of which can be found on the right/button mouse menu.

Properties

[Text](#)

Edit operations

[Copy](#)
[Cut](#)

Dynamic table operations

[AppendRow](#)
[DeleteRow](#)

Delete	DuplicateRow
IsCopyEnabled	InsertRow
IsCutEnabled	IsFirstRow
IsDeleteEnabled	IsIndynamicTable
IsPasteEnabled	IsLastRow
Paste	MoveRowDown
	MoveRowUp

The following methods provide the functionality of the Authentic entry helper windows for range objects.

Operations of the entry helper windows

Elements

[CanPerformActionWithin](#)
[CanPerformAction](#)
[PerformAction](#)

Attributes

[GetElementAttributeValue](#)
[GetElementAttributeNames](#)
[GetElementHierarchy](#)
[HasElementAttribute](#)
[IsTextStateApplied](#)
[SetElementAttributeValue](#)

Entities

[GetEntityNames](#)
[InsertEntity](#)

Description

[AuthenticRange](#) objects are the 'cursor' selections of the automation interface. You can use them to point to any cursor position in the Authentic view, or select a portion of the document. The operations available for [AuthenticRange](#) objects then work on this selection in the same way, as the corresponding operations of the user interface do with the current user interface selection. The main difference is that you can use an arbitrary number of [AuthenticRange](#) objects at the same time, whereas there is exactly one cursor selection in the user interface.

To get to an initial range object use [AuthenticView.Selection](#), to obtain a range corresponding with the current cursor selection in the user interface. Alternatively, some trivial ranges are accessible via the read/only properties [AuthenticView.DocumentBegin](#), [AuthenticView.DocumentEnd](#), and [AuthenticView.WholeDocument](#). The most flexible method is [AuthenticView.Goto](#), which allows navigation to a specific portion of the document within one call. For more complex selections, combine the above, with the various navigation methods on range objects listed in the first table on this page.

Another method to select a portion of the document is to use the position properties of the range object. Two positioning systems are available and can be combined arbitrarily:

- **Absolute** text cursor positions, starting with position 0 at the document beginning, can be set and retrieved for the beginning and end of a range. For more information see [FirstTextPosition](#) and [LastTextPosition](#). This method requires complex internal calculations and should be used with care.
- The **XMLData** element and a text position inside this element, can be set and retrieved for the beginning and end of a range. For more information see [FirstXMLData](#), [FirstXMLDataOffset](#), [LastXMLData](#), and [LastXMLDataOffset](#). This method is very efficient but requires knowledge on the underlying document structure. It can be used to locate XMLData objects and perform operations on them otherwise not accessible through the user interface.

Modifications to the document content can be achieved by various methods:

- The [Text](#) property allows you to retrieve the document text selected by the range object. If set, the selected document text gets replaced with the new text.
- The standard document edit functions [Cut](#), [Copy](#), [Paste](#) and [Delete](#).

- Table operations for tables that can grow dynamically.
- Methods that map the functionality of the Authentic entry helper windows.
- Access to the [XMLData](#) objects of the underlying document to modify them directly.

AppendRow

See also

Method: [AppendRow\(\)](#) as Boolean

Description

If the beginning of the range is inside a dynamic table, this method inserts a new row at the end of the selected table. The selection of the range is modified to point to the beginning of the new row. The function returns *true* if the append operation was successful, otherwise *false*.

Errors

- 2001 The authentic range object or its related view object is no longer valid.
 2005 Invalid address for the return parameter was specified.

Examples

```
' -----
' XMLSpy scripting environment - VBScript
' Append row at end of current dynamically growable table
' -----
Dim objRange
' we assume that the active document is open in authentic view mode
Set objRange = Application.ActiveDocument.AuthenticView.Selection

' check if we can insert something
If objRange.IsInDynamicTable Then
    objRange.AppendRow
    ' objRange points to beginning of new row
    objRange.Select
End If
```

Application

See also

Property: [Application](#) as [Application](#) (read-only)

Description

Accesses the XMLSpy application object.

Errors

- 2001 The authentic range object or its related view object is no longer valid.
 2005 Invalid address for the return parameter was specified.

CanPerformAction

See also

Method: [CanPerformAction](#) (*eAction* as [SPYAuthenticActions](#), *strElementName* as String) as Boolean

Description

[CanPerformAction](#) and its related methods enable access to the entry-helper functions of

Authentic. This function allows easy and consistent modification of the document content, without having to know exactly where the modification will take place. The beginning of the range object is used to locate the next valid location where the specified action can be performed. If the location can be found, the method returns *True*, otherwise it returns *False*.

HINT: To find out all valid element names for a given action, use [CanPerformActionWith](#).

Errors

- 2001 The authentic range object or its related view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.
- 2007 Invalid action was specified.

Examples

See [PerformAction](#).

CanPerformActionWith

See also

Method: [CanPerformActionWith](#) (*eAction* as [SPYAuthenticActions](#),
out_arrElementNames as Variant)

Description

[PerformActionWith](#) and its related methods, enable access to the entry-helper functions of Authentic. These function allows easy and consistent modification of the document content without having to know exactly where the modification will take place.

This method returns an array of those element names that the specified action can be performed with.

HINT: To apply the action use [CanPerformActionWith](#).

Errors

- 2001 The authentic range object, or its related view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.
- 2007 Invalid action was specified.

Examples

See [PerformAction](#).

Clone

See also

Method: [Clone\(\)](#) as [AuthenticRange](#)

Description

Returns a copy of the range object.

Errors

- 2001 The authentic range object, or its related view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.

CollapsToBegin

See also

Method: [CollapsToBegin\(\)](#) as [AuthenticRange](#)

Description

Sets the end of the range object to its begin. The method returns the modified range object.

Errors

- 2001 The authentic range object, or its related view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.

CollapsToEnd

See also

Method: [CollapsToEnd\(\)](#) as [AuthenticRange](#)

Description

Sets the beginning of the range object to its end. The method returns the modified range object.

Errors

- 2001 The authentic range object, or its related view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.

Copy

See also

Method: [Copy\(\)](#) as Boolean

Description

Returns *False* if the range contains no portions of the document that may be copied. Returns *True* if text, and in case of fully selected XML elements the elements as well, has been copied to the copy/paste buffer.

Errors

- 2001 The authentic range object or its related view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.

Cut

See also

Method: [Cut\(\)](#) as Boolean

Description

Returns *False* if the range contains portions of the document that may not be deleted. Returns *True* after text, and in case of fully selected XML elements the elements as well, has been deleted from the document and saved in the copy/paste buffer.

Errors

- 2001 The authentic range object, or its related view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.

Delete

See also

Method: `Delete()` as Boolean

Description

Returns *False* if the range contains portions of the document that may not be deleted. Returns *True* after text, and in case of fully selected XML elements the elements as well, has been deleted from the document.

Errors

- 2001 The authentic range object or its related view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.

DeleteRow

See also

Method: `DeleteRow()` as Boolean

Description

If the beginning of the range is inside a dynamic table, this method deletes the selected row. The selection of the range gets modified to point to the next element after the deleted row. The function returns *true*, if the delete operation was successful, otherwise *false*.

Errors

- 2001 The authentic range object, or its related view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.

Examples

```
' -----  
' XMLSpy scripting environment - VBScript  
' Delete selected row from dynamically growing table  
' -----  
Dim objRange  
' we assume that the active document is open in authentic view mode  
Set objRange = Application.ActiveDocument.AuthenticView.Selection  
  
' check if we are in a table  
If objRange.IsInDynamicTable Then  
    objRange.DeleteRow  
End If
```

DuplicateRow

See also

Method: `DuplicateRow()` as Boolean

Description

If the beginning of the range is inside a dynamic table, this method inserts a duplicate of the current row after the selected one. The selection of the range gets modified to point to the

beginning of the new row. The function returns *true* if the duplicate operation was successful, otherwise *false*.

Errors

- 2001 The authentic range object, or its related view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.

Examples

```
' -----
' XMLSpy scripting environment - VBScript
' duplicate row in current dynamically growable table
' -----
Dim objRange
' we assume that the active document is open in authentic view mode
Set objRange = Application.ActiveDocument.AuthenticView.Selection

' check if we can insert something
If objRange.IsInDynamicTable Then
    objRange.DuplicateRow
    ' objRange points to beginning of new row
    objRange.Select
End If
```

EvaluateXPath

Method: [EvaluateXPath](#) (strExpression as string) strValue as string

Return Value

The method returns a string

Description

[EvaluateXPath\(\)](#) executes an XPath expressions with the context node being the beginning of the range selection. The result is returned as string, in the case of a sequence it is a space-separated string. If XML context node is irrelevant, the user may provide any node, like [AuthenticView.XMLDataRoot](#).

Errors

- 2001 Invalid object
- 2005 Invalid parameter
- 2008 Internal error
- 2202 Missing context node
- 2211 XPath error

ExpandTo

See also

Method: [ExpandTo](#) (*eKind* as [SPYAuthenticElementKind](#)), as [AuthenticRange](#)

Description

Selects the whole element of type *eKind*, that starts at, or contains, the first cursor position of the range. The method returns the modified range object.

Errors

- 2001 The authentic range object, or its related view object is no longer valid.
- 2003 Range expansion would be beyond end of document.
- 2005 Invalid address for the return parameter was specified.

FirstTextPosition

See also

Property: [FirstTextPosition](#) as Long

Description

Set or get the left-most text position index of the range object. This index is always less or equal to [LastTextPosition](#). Indexing starts with 0 at document beginning, and increments with every different position that the text cursor can occupy. Incrementing the text position by 1, has the same effect as the cursor-right key. Decrementing the text position by 1 has the same effect as the cursor-left key.

If you set [FirstTextPosition](#) to a value greater than the current [LastTextPosition](#), [LastTextPosition](#) gets set to the new [FirstTextPosition](#).

HINT: Use text cursor positions with care, since this is a costly operation compared to [XMLData](#) based cursor positioning.

Errors

- 2001 The authentic range object, or its related view object is not valid.
- 2005 Invalid address for the return parameter was specified.
- 2006 A text position outside the document was specified.

Examples

```
' -----
' XMLSpy scripting environment - VBScript
' -----
Dim objAuthenticView
' we assume that the active document is open in authentic view mode
Set objAuthenticView = Application.ActiveDocument.AuthenticView

nDocStartPosition = objAuthenticView.DocumentBegin.FirstTextPosition
nDocEndPosition = objAuthenticView.DocumentEnd.FirstTextPosition

' let's create a range that selects the whole document
' in an inefficient way
Dim objRange
' we need to get a (any) range object first
Set objRange = objAuthenticView.DocumentBegin
objRange.FirstTextPosition = nDocStartPosition
objRange.LastTextPosition = nDocEndPosition

' let's check if we got it right
If objRange.IsEqual(objAuthenticView.WholeDocument) Then
    MsgBox "Test using direct text cursor positioning was ok"
Else
    MsgBox "Ooops! "
End If
```

FirstXMLData

See also

Property: [FirstXMLData](#) as [XMLData](#)

Description

Set or get the first [XMLData](#) element in the underlying document that is partially, or completely selected by the range. The exact beginning of the selection is defined by the

[FirstXMLDataOffset](#) attribute.

Whenever you set `FirstXMLData` to a new data object, [FirstXMLDataOffset](#) gets set to the first cursor position inside this element. Only XMLData objects that have a cursor position may be used. If you set `FirstXMLData` / [FirstXMLDataOffset](#) selects a position greater then the current [LastXMLData](#) / [LastXMLDataOffset](#), the latter gets moved to the new start position.

HINT: You can use the [FirstXMLData](#) and [LastXMLData](#) properties, to directly access and manipulate the underlying XML document in those cases where the methods available with the [AuthenticRange](#) object are not sufficient.

Errors

- 2001 The authentic range object, or its related view object is not valid.
- 2005 Invalid address for the return parameter was specified.
- 2008 Internal error
- 2009 The XMLData object cannot be accessed.

Examples

```
' -----
' XMLSpy scripting environment - VBScript
' show name of currently selected XMLData element
' -----
Dim objAuthenticView
' we assume that the active document is open in authentic view mode
Set objAuthenticView = Application.ActiveDocument.AuthenticView

Dim objXMLData
Set objXMLData = objAuthenticView.Selection.FirstXMLData
' authentic view adds a 'text' child element to elements
' of the document which have content. So we have to go one
' element up.
Set objXMLData = objXMLData.Parent
MsgBox "Current selection selects element " & objXMLData.Name
```

FirstXMLDataOffset

See also

Property: [FirstXMLDataOffset](#) as Long

Description

Set or get the cursor position offset inside [FirstXMLData](#) element for the beginning of the range. Offset positions are based on the characters returned by the [Text](#) property, and start with 0. When setting a new offset, use -1 to set the offset to the last possible position in the element. The following cases require specific attention:

- The textual form of entries in Combo Boxes, Check Boxes and similar controls can be different from what you see on screen. Although the data offset is based on this text, there only two valid offset positions, one at the beginning and one at the end of the entry. An attempt to set the offset to somewhere in the middle of the entry, will result in the offset being set to the end.
- The textual form of XML Entities might differ in length from their representation on the screen. The offset is based on this textual form.

If `FirstXMLData` / [FirstXMLDataOffset](#) selects a position after the current [LastXMLData](#) / [LastXMLDataOffset](#), the latter gets moved to the new start position.

Errors

- 2001 The authentic range object, or its related view object is not valid.
 2005 Invalid offset was specified.
 Invalid address for the return parameter was specified.

Examples

```

' -----
' XMLSpy scripting environment - VBScript
' Select the complete text of an XMLData element
' using XMLData based selection and ExpandTo
' -----
Dim objAuthenticView
' we assume that the active document is open in authentic view mode
Set objAuthenticView = Application.ActiveDocument.AuthenticView

' first we use the XMLData based range properties
' to select all text of the first XMLData element
' in the current selection
Dim objRange
Set objRange = objAuthenticView.Selection
objRange.FirstXMLDataOffset = 0 ' start at beginning of element text
objRange.LastXMLData = objRange.FirstXMLData ' select only one element
objRange.LastXMLDataOffset = -1 ' select till its end

' the same can be achieved with the ExpandTo method
Dim objRange2
Set objRange2 = objAuthenticView.Selection.ExpandTo(spyAuthenticTag)

' were we successful?
If objRange.IsEqual(objRange2) Then
    objRange.Select()
Else
    MsgBox "Oops"
End If
  
```

GetElementAttributeNames

See also

Method: `GetElementAttributeNames` (*strElementName* as String,
out_arrAttributeNames as Variant)

Description

Retrieve the names of all attributes for the enclosing element with the specified name. Use the element/attribute pairs, to set or get the attribute value with the methods [GetElementAttributeValue](#) and [SetElementAttributeValue](#).

Errors

- 2001 The authentic range object, or its related view object is no longer valid.
 2005 Invalid element name was specified.
 Invalid address for the return parameter was specified.

Examples

See [SetElementAttributeValue](#).

GetElementAttributeValue

See also

Method: `GetElementAttributeValue` (*strElementName* as String, *strAttributeName*
 as String) as String

Description

Retrieve the value of the attribute specified in `strAttributeName`, for the element identified with `strElementName`. If the attribute is supported but has no value assigned, the empty string is returned. To find out the names of attributes supported by an element, use [GetElementAttributeNames](#), or [HasElementAttribute](#).

Errors

- 2001 The authentic range object, or its related view object is no longer valid.
- 2005 Invalid element name was specified.
Invalid attribute name was specified.
Invalid address for the return parameter was specified.

Examples

See [SetElementAttributeValue](#).

GetElementHierarchy**See also**

Method: [GetElementHierarchy](#) (*out_arrElementNames* as Variant)

Description

Retrieve the names of all XML elements that are parents of the current selection. Inner elements get listed before enclosing elements. An empty list is returned whenever the current selection is not inside a single `XMLData` element.

The names of the element hierarchy, together with the range object uniquely identify `XMLData` elements in the document. The attributes of these elements can be directly accessed by [GetElementAttributeNames](#), and related methods.

Errors

- 2001 The authentic range object, or its related view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.

C# Examples

```

' -----
' C#
' -----

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            XMLSpyLib.Application app = new XMLSpyLib.Application();

            app.ShowApplication(true);

            XMLSpyLib.AuthenticView view = app.ActiveDocument.AuthenticView;
            XMLSpyLib.AuthenticRange range = view.DocumentBegin;

            object o = null;
            range.GetElementHierarchy(ref o);

            object[] elements = (object[])o;
        }
    }
}

```

```

        foreach (string e in elements)
        {
            Console.WriteLine(e);
        }
    }
}

```

Also see: [SetElementAttributeValue](#).

GetEntityNames

See also

Method: [GetEntityNames](#) (*out_arrEntityNames* as Variant)

Description

Retrieve the names of all defined entities. The list of retrieved entities is independent of the current selection, or location. Use one of these names with the [InsertEntity](#) function.

Errors

- 2001 The authentic range object, or its related view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.

Examples

See: [GetElementHierarchy](#) and [InsertEntity](#).

GetVariableValue

Method: [GetVariableValue](#)(*strName* as string) *strVal* as string

Return Value

Gets the value of the variable named as the method's parameter.

Errors

- 2001 Invalid object.
- 2202 No context.
- 2204 No such variable in scope
- 2205 Variable cannot be evaluated
- 2206 Variable returns sequence
- 2209 Invalid parameter

Goto

See also

Method: [Goto](#) (*eKind* as [SPYAuthenticElementKind](#), *nCount* as Long, *eFrom* as [SPYAuthenticDocumentPosition](#)) as [AuthenticRange](#)

Description

Sets the range to point to the beginning of the *nCount* element of type *eKind* . The start position is defined by the parameter *eFrom* .

Use positive values for `nCount` to navigate to the document end. Use negative values to navigate to the beginning of the document. The method returns the modified range object.

Errors

- 2001 The authentic range object, or its related view object is no longer valid.
- 2003 Target lies after end of document.
- 2004 Target lies before begin of document.
- 2005 Invalid element kind specified.
Invalid start position specified.
Invalid address for the return parameter was specified.

GotoNext

See also

Method: `GotoNext` (`eKind` as [SPYAuthenticElementKind](#)) as [AuthenticRange](#)

Description

Sets the range to the beginning of the next element of type `eKind`. The method returns the modified range object.

Errors

- 2001 The authentic range object, or its related view object is no longer valid.
- 2003 Target lies after end of document.
- 2005 Invalid element kind specified.
Invalid address for the return parameter was specified.

Examples

```
' -----
' XMLSpy scripting environment - VBScript
' Scan through the whole document word-by-word
' -----
Dim objAuthenticView
' we assume that the active document is open in authentic view mode
Set objAuthenticView = Application.ActiveDocument.AuthenticView

Dim objRange
Set objRange = objAuthenticView.DocumentBegin
Dim bEndOfDocument
bEndOfDocument = False

On Error Resume Next
While Not bEndOfDocument
    objRange.GotoNext(spyAuthenticWord).Select
    If ((Err.number - vbObjectError) = 2003) Then
        bEndOfDocument = True
        Err.Clear
    ElseIf (Err.number <> 0) Then
        Err.Raise ' forward error
    End If
Wend
```

GotoNextCursorPosition

See also

Method: `GotoNextCursorPosition()` as [AuthenticRange](#)

Description

Sets the range to the next cursor position after its current end position. Returns the modified object.

Errors

- 2001 The authentic range object, or its related view object is no longer valid.
- 2003 Target lies after end of document.
- 2005 Invalid address for the return parameter was specified.

GotoPrevious

See also

Method: `GotoPrevious` (*eKind* as [SPYAuthenticElementKind](#)) as [AuthenticRange](#)

Description

Sets the range to the beginning of the element of type *eKind* which is before the beginning of the current range. The method returns the modified range object.

Errors

- 2001 The authentic range object, or its related view object is no longer valid.
- 2004 Target lies before beginning of document.
- 2005 Invalid element kind specified.
Invalid address for the return parameter was specified.

Examples

```
' -----
' XMLSpy scripting environment - VBScript
' Scan through the whole document tag-by-tag
' -----
Dim objAuthenticView
' we assume that the active document is open in authentic view mode
Set objAuthenticView = Application.ActiveDocument.AuthenticView

Dim objRange
Set objRange = objAuthenticView.DocumentEnd
Dim bEndOfDocument
bBeginOfDocument = False

On Error Resume Next
While Not bBeginOfDocument
    objRange.GotoPrevious(spyAuthenticTag).Select
    If ((Err.number - vbObjectError) = 2004) Then
        bBeginOfDocument = True
        Err.Clear
    ElseIf (Err.number <> 0) Then
        Err.Raise ' forward error
    End If
Wend
```

GotoPreviousCursorPosition

See also

Method: `GotoPreviousCursorPosition()` as [AuthenticRange](#)

Description

Set the range to the cursor position immediately before the current position. Returns the modified object.

Errors

- 2001 The authentic range object, or its related view object is no longer valid.
- 2004 Target lies before begin of document.
- 2005 Invalid address for the return parameter was specified.

HasElementAttribute**See also**

Method: `HasElementAttribute` (*strElementName* as String, *strAttributeName* as String) as Boolean

Description

Tests if the enclosing element with name *strElementName* , supports the attribute specified in *strAttributeName* .

Errors

- 2001 The authentic range object, or its related view object is no longer valid.
- 2005 Invalid element name was specified.
Invalid address for the return parameter was specified.

InsertEntity**See also**

Method: `InsertEntity` (*strEntityName* as String)

Description

Replace the ranges selection with the specified entity. The specified entity must be one of the entity names returned by [GetEntityNames](#).

Errors

- 2001 The authentic range object, or its related view object is no longer valid.
- 2005 Unknown entry name was specified.

Examples

```
' -----
' XMLSpy scripting environment - VBScript
' Insert the first entity in the list of available entities
' -----
Dim objRange
' we assume that the active document is open in authentic view mode
Set objRange = Application.ActiveDocument.AuthenticView.Selection

' first we get the names of all available entities as they
' are shown in the entry helper of XMLSpy
Dim arrEntities
objRange.GetEntityNames arrEntities

' we insert the first one of the list
If UBound(arrEntities) >= 0 Then
    objRange.InsertEntity arrEntities(0)
Else
    MsgBox "Sorry, no entities are available for this document"
End If
```


InsertRow

See also

Method: `InsertRow()` as Boolean

Description

If the beginning of the range is inside a dynamic table, this method inserts a new row before the current one. The selection of the range, gets modified to point to the beginning of the newly inserted row. The function returns *true* if the insert operation was successful, otherwise *false*.

Errors

- 2001 The authentic range object, or its related view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.

Examples

```
'-----  
' XMLSpy scripting environment - VBScript  
' Insert row at beginning of current dynamically growing table  
'-----  
  
Dim objRange  
' we assume that the active document is open in authentic view mode  
Set objRange = Application.ActiveDocument.AuthenticView.Selection  
  
' check if we can insert something  
If objRange.IsInDynamicTable Then  
    objRange.InsertRow  
    ' objRange points to beginning of new row  
    objRange.Select  
End If
```

IsCopyEnabled

See also

Property: `IsCopyEnabled` as Boolean (read-only)

Description

Checks if the copy operation is supported for this range.

Errors

- 2001 The authentic range object, or its related view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.

IsCutEnabled

See also

Property: `IsCutEnabled` as Boolean (read-only)

Description

Checks if the cut operation is supported for this range.

Errors

- 2001 The authentic range object, or its related view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.

IsDeleteEnabled

See also

Property: [IsDeleteEnabled](#) as Boolean (read-only)

Description

Checks if the delete operation is supported for this range.

Errors

- 2001 The authentic range object, or its related view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.

IsEmpty

See also

Method: [IsEmpty\(\)](#) as Boolean

Description

Tests if the first and last position of the range are equal.

Errors

- 2001 The authentic range object, or its related view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.

IsEqual

See also

Method: [IsEqual](#) ([objCmpRange](#) as [AuthenticRange](#)) as Boolean

Description

Tests if the start and end of both ranges are the same.

Errors

- 2001 One of the two range objects being compared, is invalid.
- 2005 Invalid address for a return parameter was specified.

IsFirstRow

See also

Property: [IsFirstRow](#) as Boolean (read-only)

Description

Test if the range is in the first row of a table. Which table is taken into consideration depends on the extend of the range. If the selection exceeds a single row of a table, the check is if this table is the first element in an embedding table. See the entry helpers of the user manual for more information.

Errors

- 2001 The authentic range object, or its related view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.

IsInDynamicTable

See also

Method: `IsInDynamicTable()` as Boolean

Description

Test if the whole range is inside a table that supports the different row operations like 'insert', 'append', duplicate, etc.

Errors

- 2001 The authentic range object, or its related view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.

IsLastRow

See also

Property: `IsLastRow` as Boolean (read-only)

Description

Test if the range is in the last row of a table. Which table is taken into consideration depends on the extend of the range. If the selection exceeds a single row of a table, the check is if this table is the last element in an embedding table. See the entry helpers of the user manual for more information.

Errors

- 2001 The authentic range object, or its related view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.

IsPasteEnabled

See also

Property: `IsPasteEnabled` as Boolean (read-only)

Description

Checks if the paste operation is supported for this range.

Errors

- 2001 The authentic range object, or its related view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.

IsSelected

Property: `IsSelected` as Boolean

Description

Returns true() if selection is present. The selection range still can be empty: that happens when e.g. only the cursor is set.

IsTextStateApplied

See also

Method: `IsTextStateApplied` (*i_strElementName* as String) as Boolean

Description

Checks if all the selected text is embedded into an XML Element with name `i_strElementName`. Common examples for the parameter `i_strElementName` are "strong", "bold" or "italic".

Errors

- 2001 The authentic range object, or its related view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.

LastTextPosition

See also

Property: `LastTextPosition` as Long

Description

Set or get the rightmost text position index of the range object. This index is always greater or equal to [FirstTextPosition](#). Indexing starts with 0 at the document beginning, and increments with every different position that the text cursor can occupy. Incrementing the test position by 1, has the same effect as the cursor-right key. Decreasing the test position by 1 has the same effect as the cursor-left key.

If you set `LastTextPosition` to a value less then the current [FirstTextPosition](#), [FirstTextPosition](#) gets set to the new `LastTextPosition`.

HINT: Use text cursor positions with care, since this is a costly operation compared to XMLData based cursor positioning.

Errors

- 2001 The authentic range object, or its related view object is not valid.
- 2005 Invalid address for the return parameter was specified.
- 2006 A text position outside the document was specified.

Examples

```

' -----
' XMLSpy scripting environment - VBScript
' -----
Dim objAuthenticView
' we assume that the active document is open in authentic view mode
Set objAuthenticView = Application.ActiveDocument.AuthenticView

nDocStartPosition = objAuthenticView.DocumentBegin.FirstTextPosition
nDocEndPosition = objAuthenticView.DocumentEnd.FirstTextPosition

' let's create a range that selects the whole document
' in an inefficient way
Dim objRange
' we need to get a (any) range object first
Set objRange = objAuthenticView.DocumentBegin
objRange.FirstTextPosition = nDocStartPosition
objRange.LastTextPosition = nDocEndPosition

' let's check if we got it right

```

```
If objRange.isEqual(objAuthenticView.WholeDocument) Then
    MsgBox "Test using direct text cursor positioning was ok"
Else
    MsgBox "Oops! "
End If
```

LastXMLData

See also

Property: [LastXMLData](#) as [XMLData](#)

Description

Set or get the last [XMLData](#) element in the underlying document that is partially or completely selected by the range. The exact end of the selection is defined by the [LastXMLDataOffset](#) attribute.

Whenever you set [LastXMLData](#) to a new data object, [LastXMLDataOffset](#) gets set to the last cursor position inside this element. Only [XMLData](#) objects that have a cursor position may be used. If you set [LastXMLData](#) / [LastXMLDataOffset](#), select a position less than the current [FirstXMLData](#) / [FirstXMLDataOffset](#), the latter gets moved to the new end position.

HINT: You can use the [FirstXMLData](#) and [LastXMLData](#) properties to directly access and manipulate the underlying XML document in those cases, where the methods available with the [AuthenticRange](#) object are not sufficient.

Errors

- 2001 The authentic range object, or its related view object is not valid.
- 2005 Invalid address for the return parameter was specified.
- 2008 Internal error
- 2009 The [XMLData](#) object cannot be accessed.

LastXMLDataOffset

See also

Property: [LastXMLDataOffset](#) as Long

Description

Set or get the cursor position inside [LastXMLData](#) element for the end of the range.

Offset positions are based on the characters returned by the [Text](#) property and start with 0. When setting a new offset, use -1 to set the offset to the last possible position in the element. The following cases require specific attention:

- The textual form of entries in Combo Boxes, Check Boxes and similar controls can be different from what you see on the screen. Although, the data offset is based on this text, there only two valid offset positions, one at the beginning and one at the end of the entry. An attempt to set the offset to somewhere in the middle of the entry, will result in the offset being set to the end.
- The textual form of XML Entities might differ in length from their representation on the screen. The offset is based on this textual form.

If [LastXMLData](#) / [LastXMLDataOffset](#) selects a position before [FirstXMLData](#) / [FirstXMLDataOffset](#), the latter gets moved to the new end position.

Errors

- 2001 The authentic range object, or its related view object is not valid.
- 2005 Invalid offset was specified.
Invalid address for the return parameter was specified.

Examples

```

' -----
' XMLSpy scripting environment - VBScript
' Select the complete text of an XMLData element
' using XMLData based selection and ExpandTo
' -----
Dim objAuthenticView
' we assume that the active document is open in authentic view mode
Set objAuthenticView = Application.ActiveDocument.AuthenticView

' first we use the XMLData based range properties
' to select all text of the first XMLData element
' in the current selection
Dim objRange
Set objRange = objAuthenticView.Selection
objRange.FirstXMLDataOffset = 0 ' start at beginning of element text
objRange.LastXMLData = objRange.FirstXMLData ' select only one element
objRange.LastXMLDataOffset = -1 ' select till its end

' the same can be achieved with the ExpandTo method
Dim objRange2
Set objRange2 = objAuthenticView.Selection.ExpandTo(spyAuthenticTag)

' were we successful?
If objRange.IsEqual(objRange2) Then
    objRange.Select()
Else
    MsgBox "Oops"
End If

```

MoveBegin**See also**

Method: `MoveBegin` (*eKind* as [SPYAuthenticElementKind](#), *nCount* as Long) as [AuthenticRange](#)

Description

Move the beginning of the range to the beginning of the *nCount* element of type *eKind*. Counting starts at the current beginning of the range object.

Use positive numbers for *nCount* to move towards the document end, use negative numbers to move towards document beginning. The end of the range stays unmoved, unless the new beginning would be larger than it. In this case, the end is moved to the new beginning. The method returns the modified range object.

Errors

- 2001 The authentic range object, or its related view object is no longer valid.
- 2003 Target lies after end of document.
- 2004 Target lies before beginning of document.
- 2005 Invalid element kind specified.
Invalid address for the return parameter was specified.

MoveEnd

See also

Method: `MoveEnd` (*eKind* as [SPYAuthenticElementKind](#), *nCount* as Long) as [AuthenticRange](#)

Description

Move the end of the range to the begin of the *nCount* element of type *eKind*. Counting starts at the current end of the range object.

Use positive numbers for *nCount* to move towards the document end, use negative numbers to move towards document beginning. The beginning of the range stays unmoved, unless the new end would be less than it. In this case, the beginning gets moved to the new end. The method returns the modified range object.

Errors

- 2001 The authentic range object, or its related view object is no longer valid.
- 2003 Target lies after end of document.
- 2004 Target lies before begin of document.
- 2005 Invalid element kind specified.
Invalid address for the return parameter was specified.

MoveRowDown

See also

Method: `MoveRowDown`() as Boolean

Description

If the beginning of the range is inside a dynamic table and selects a row which is not the last row in this table, this method swaps this row with the row immediately below. The selection of the range moves with the row, but does not otherwise change. The function returns *true* if the move operation was successful, otherwise *false*.

Errors

- 2001 The authentic range object or its related view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.

MoveRowUp

See also

Method: `MoveRowUp`() as Boolean

Description

If the beginning of the range is inside a dynamic table and selects a row which is not the first row in this table, this method swaps this row with the row above. The selection of the range moves with the row, but does not change otherwise. The function returns *true* if the move operation was successful, otherwise *false*.

Errors

- 2001 The authentic range object, or its related view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.

Examples

See [JScript - Bubble Sort Dynamic Tables](#).

Parent**See also**

Property: `Parent` as [AuthenticView](#) (read-only)

Description

Access the view that owns this range object.

Errors

- 2001 The authentic range object, or its related view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.

Paste**See also**

Method: `Paste()` as Boolean

Description

Returns *False* if the copy/paste buffer is empty, or its content cannot replace the current selection.

Otherwise, deletes the current selection, inserts the content of the copy/paste buffer, and returns *True*.

Errors

- 2001 The authentic range object, or its related view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.

PerformAction**See also**

Method: `PerformAction` (`eAction` as [SPYAuthenticActions](#), `strElementName` as String) as Boolean

Description

`PerformAction` and its related methods, give access to the entry-helper functions of Authentic. This function allows easy and consistent modification of the document content without a need to know exactly where the modification will take place. The beginning of the range object is used to locate the next valid location where the specified action can be performed. If no such location can be found, the method returns *False*. Otherwise, the document gets modified and the range points to the beginning of the modification.

HINT: To find out element names that can be passed as the second parameter use [CanPerformActionWith](#).

Errors

- 2001 The authentic range object, or its related view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.
- 2007 Invalid action was specified.

Examples

```

' -----
' XMLSpy scripting environment - VBScript
' Insert the innermost element
' -----
Dim objRange
' we assume that the active document is open in authentic view mode
Set objRange = Application.ActiveDocument.AuthenticView.Selection

' we determine the elements that can be inserted at the current position
Dim arrElements()
objRange.CanPerformActionWith spyAuthenticInsertBefore, arrElements

' we insert the first (innermost) element
If UBound(arrElements) >= 0 Then
    objRange.PerformAction spyAuthenticInsertBefore, arrElements(0)
    ' objRange now points to the beginning of the inserted element
    ' we set a default value and position at its end
    objRange.Text = "Hello"
    objRange.ExpandTo(spyAuthenticTag).CollapsToEnd().Select
Else
    MsgBox "Can't insert any elements at current position"
End If

```

Select**See also****Method:** [Select\(\)](#)**Description**

Makes this range the current user interface selection. You can achieve the same result using: '
objRange.Parent.Selection = objRange

Errors

2001 The authentic range object or its related view object is no longer valid.

Examples

```

' -----
' XMLSpy scripting environment - VBScript
' -----
Dim objAuthenticView
' we assume that the active document is open in authentic view mode
Set objAuthenticView = Application.ActiveDocument.AuthenticView

' set current selection to end of document
objAuthenticView.DocumentEnd.Select()

```

SelectNext**See also****Method:** [SelectNext](#) (*eKind* as [SPYAuthenticElementKind](#)) as [AuthenticRange](#)**Description**

Selects the element of type *eKind* after the current end of the range. The method returns the modified range object.

Errors

2001 The authentic range object, or its related view object is no longer valid.

- 2003 Target lies after end of document.
- 2005 Invalid element kind specified.
Invalid address for the return parameter was specified.

Examples

```

' -----
' XMLSpy scripting environment - VBScript
' Scan through the whole document word-by-word
' -----
Dim objAuthenticView
' we assume that the active document is open in authentic view mode
Set objAuthenticView = Application.ActiveDocument.AuthenticView

Dim objRange
Set objRange = objAuthenticView.DocumentBegin
Dim bEndOfDocument
bEndOfDocument = False

On Error Resume Next
While Not bEndOfDocument
    objRange.SelectNext(spyAuthenticWord).Select
    If ((Err.number - vbObjectError) = 2003) Then
        bEndOfDocument = True
        Err.Clear
    ElseIf (Err.number <> 0) Then
        Err.Raise ' forward error
    End If
Wend

```

SelectPrevious**See also**

Method: [GotoPrevious](#) (*eKind* as [SPYAuthenticElementKind](#)) as [AuthenticRange](#)

Description

Selects the element of type *eKind* before the current beginning of the range. The method returns the modified range object.

Errors

- 2001 The authentic range object, or its related view object is no longer valid.
- 2004 Target lies before begin of document.
- 2005 Invalid element kind specified.
Invalid address for the return parameter was specified.

Examples

```

' -----
' XMLSpy scripting environment - VBScript
' Scan through the whole document tag-by-tag
' -----
Dim objAuthenticView
' we assume that the active document is open in authentic view mode
Set objAuthenticView = Application.ActiveDocument.AuthenticView

Dim objRange
Set objRange = objAuthenticView.DocumentEnd
Dim bEndOfDocument
bBeginOfDocument = False

On Error Resume Next

```

```

While Not bBeginOfDocument
    objRange.SelectPrevious(spyAuthenticTag).Select
    If ((Err.number - vbObjecterror) = 2004) Then
        bBeginOfDocument = True
        Err.Clear
    ElseIf (Err.number <> 0) Then
        Err.Raise ' forward error
    End If
Wend

```

SetElementAttributeValue

See also

Method: `SetElementAttributeValue` (*strElementName* as String, *strAttributeName* as String, *strAttributeValue* as String)

Description

Retrieve the value of the attribute specified in *strAttributeName* for the element identified with *strElementName*. If the attribute is supported but has no value assigned, the empty string is returned. To find out the names of attributes supported by an element, use [GetElementAttributeNames](#), or [HasElementAttribute](#).

Errors

- 2001 The authentic range object or its related view object is no longer valid.
- 2005 Invalid element name was specified.
Invalid attribute name was specified.
Invalid attribute value was specified.

Examples

```

' -----
' XMLSpy scripting environment - VBScript
' Get and set element attributes
' -----

Dim objRange
' we assume that the active document is open in authentic view mode
Set objRange = Application.ActiveDocument.AuthenticView.Selection

' first we find out all the elements below the beginning of the range
Dim arrElements
objRange.GetElementHierarchy arrElements

If IsArray(arrElements) Then
    If UBound(arrElements) >= 0 Then
        ' we use the top level element and find out its valid attributes
        Dim arrAttrs()
        objRange.GetElementAttributeNames arrElements(0), arrAttrs

        If UBound(arrAttrs) >= 0 Then
            ' we retrieve the current value of the first valid
            attribute
            Dim strAttrVal
            strAttrVal = objRange.GetElementAttributeValue
            (arrElements(0), arrAttrs(0))
            msgbox "current value of " & arrElements(0) & "/" &
            arrAttrs(0) & " is: " & strAttrVal

            ' we change this value and read it again
            strAttrVal = "Hello"
            objRange.SetElementAttributeValue arrElements(0),
            arrAttrs(0), strAttrVal
        End If
    End If
End If

```

```

        strAttrVal = objRange.GetElementAttributeValue
(arrElements(0), arrAttrs(0))
        msgbox "new value of " & arrElements(0) & "/" &
arrAttrs(0) & " is: " & strAttrVal
    End If
End If
End If

```

SetFromRange

See also

Method: `SetFromRange` (*objSrcRange* as [AuthenticRange](#))

Description

Sets the range object to the same beginning and end positions as `objSrcRange`.

Errors

- 2001 One of the two range objects, is invalid.
- 2005 Null object was specified as source object.

SetVariableValue

Method: `SetVariableValue`(*strName* as string, *strValue* as string)

Return Value

Sets the value (second parameter) of the variable named in the first parameter.

Errors

- 2201 Invalid object.
- 2202 No context.
- 2204 No such variable in scope
- 2205 Variable cannot be evaluated
- 2206 Variable returns sequence
- 2207 Variable read-only
- 2208 No modification allowed

Text

See also

Property: `Text` as String

Description

Set or get the textual content selected by the range object.

The number of characters retrieved are not necessarily identical, as there are text cursor positions between the beginning and end of the selected range. Most document elements support an end cursor position different to the beginning cursor position of the following element. Drop-down lists maintain only one cursor position, but can select strings of any length. In the case of radio buttons and check boxes, the text property value holds the string of the corresponding XML element.

If the range selects more than one element, the text is the concatenation of the single texts. XML entities are expanded so that '&' is expected as '&'.

Setting the text to the empty string, does not delete any XML elements. Use [Cut](#), [Delete](#) or [PerformAction](#) instead.

Errors

- 2001 The authentic range object or its related view object is no longer valid.
- 2005 Invalid address for a return parameter was specified.

3.2.6 AuthenticView

See also

Properties

[Application](#)
[AsXMLString](#)
[DocumentBegin](#)
[DocumentEnd](#)
[Event](#)
[MarkupStyle](#)
[Parent](#)
[Section](#)
[XMLDataRoot](#)
[WholeDocument](#)

Methods

[Goto](#)
[IsRedoEnabled](#)
[IsUndoEnabled](#)
[Print](#)
[Redo](#)
[Undo](#)
[UpdateXMLInstanceEntities](#)

Events

[OnBeforeCopy](#)
[OnBeforeCut](#)
[OnBeforeDelete](#)
[OnBeforeDrop](#)
[OnBeforePaste](#)
[OnDragOver](#)
[OnKeyboardEvent](#)
[OnMouseEvent](#)
[OnSelectionChanged](#)

Description

AuthenticView and its child objects [AuthenticRange](#) and [AuthenticDataTransfer](#) provide you with an interface for Authentic View, which allow easy and consistent modification of document contents. These interfaces replace the following interfaces which are marked now as **obsolete**:

[OldAuthenticView](#) (old name was DocEditView)
[AuthenticSelection](#) (old name was DocEditSelection, superseded by [AuthenticRange](#))
[AuthenticEvent](#) (old name was DocEditEvent)

Interfaces

AuthenticView gives you easy access to specific features such as printing, the multi-level undo buffer, and the current cursor selection, or position.

AuthenticView uses objects of type [AuthenticRange](#) to make navigation inside the document straight-forward, and to allow for the flexible selection of logical text elements. Use the properties [DocumentBegin](#), [DocumentEnd](#), or [WholeDocument](#) for simple selections, while using the [Goto](#) method for more complex selections. To navigate relative to a given document range, see the methods and properties of the [AuthenticRange](#) object.

Examples

```
' -----
' XMLSpy scripting environment - VBScript
' secure access to authentic view object
' -----
Dim objDocument
Set objDocument = Application.ActiveDocument
If (Not objDocument Is Nothing) Then
    ' we have an active document, now check for view mode
    If (objDocument.CurrentViewMode <> spyViewAuthentic) Then
        If (Not objDocument.SwitchViewMode (spyViewAuthentic)) Then
```

```

mode"
        MsgBox "Active document does not support authentic view"
    Else
        ' now it is safe to access the authentic view object
        Dim objAuthenticView
        Set objAuthenticView = objDocument.AuthenticView
        ' now use the authentic view object

    End If
End If
Else
    MsgBox "No document is open"
End If

```

Events

OnBeforeCopy

See also

Event: *OnBeforeCopy()* as Boolean

XMLSpy scripting environment - VBScript:

```

Function On_AuthenticBeforeCopy()
    'On_AuthenticBeforeCopy=False 'to disable operation
EndFunction

```

XMLSpy scripting environment - JScript:

```

function On_AuthenticBeforeCopy()
{
    //return False; 'to disable operation;
}

```

XMLSpy IDE Plugin:

```

IXMLSpyPlugin.OnEvent (21, ...) //nEvent=21

```

Description

This event gets triggered before a copy operation gets performed on the document. Return *True* (or nothing) to allow copy operation. Return *False* to disable copying.

OnBeforeCut

See also

Event: *OnBeforeCut()* as Boolean

XMLSpy scripting environment - VBScript:

```

Function On_AuthenticBeforeCut()
    'On_AuthenticBeforeCut=False 'to disable operation
EndFunction

```

XMLSpy scripting environment - JScript:

```

function On_AuthenticBeforeCut()
{
    //return False; 'to disable operation;
}

```

XMLSpy IDE Plugin:

```
IXMLSpyPlugin.OnEvent (20, ...) //nEventId=20
```

Description

This event gets triggered before a cut operation gets performed on the document. Return *True* (or nothing) to allow cut operation. Return *False* to disable operation.

OnBeforeDelete**See also**

Event: `OnBeforeDelete()` as Boolean

XMLSpy scripting environment - VBScript:

```
Function On_AuthenticBeforeDelete()
    'On_AuthenticBeforeDelete=False 'to disable operation
EndFunction
```

XMLSpy scripting environment - JScript:

```
function On_AuthenticBeforeDelete()
{
    //enable/disable operation
}
```

XMLSpy IDE Plugin:

```
IXMLSpyPlugin.OnEvent (22, ...) //nEventId=22
```

Description

This event gets triggered before a delete operation gets performed on the document. Return *True* (or nothing) to allow delete operation. Return *False* to disable operation.

OnBeforeDrop**See also**

Event: `OnBeforeDrop ( i_nXPos as Long, i_nYPos as Long, i_ipRange as AuthenticRange, i_ipData as cancel Boolean`

XMLSpy scripting environment - VBScript:

```
Function On_AuthenticBeforeDrop(nXPos, nYPos, objRange, objData)
    'On_AuthenticBeforeDrop=False 'to disable operation
EndFunction
```

XMLSpy scripting environment - JScript:

```
function On_AuthenticBeforeDrop(nXPos, nYPos, objRange, objData)
{
    //enable/disable operation
}
```

XMLSpy IDE Plugin:

```
IXMLSpyPlugin.OnEvent (11, ...) //nEventId=11
```

Description

This event gets triggered whenever a previously dragged object gets dropped inside the

application window. All event related information gets passed as parameters.

The first two parameters specify the mouse position at the time when the event occurred. The parameter *objRange* passes a range object that selects the XML element below the mouse position. The value of this parameter might be *NULL*. Be sure to check before you access the range object. The parameter *objData* allows to access information about the object being dragged.

Return *False* to cancel the drop operation. Return *True* (or nothing) to continue normal operation.

Examples

```
'
'-----
' VB code snippet - connecting to object level events
'-----
' access XMLSpy (without checking for any errors)
Dim objSpy As XMLSpyLib.Application
Set objSpy = GetObject("", "XMLSpy.Application")

' this is the event callback routine connected to the OnBeforeDrop
' event of object objView
Private Function objView_OnBeforeDrop( ByVal i_nXPos As Long, ByVal i_nYPos
As Long,
                                     ByVal i_ipRange As IAuthenticRange,
                                     ByVal i_ipData As
IAuthenticDataTransfer) As Boolean

    If (Not i_ipRange Is Nothing) Then
        MsgBox ("Dropping on content is prohibited");
        Return False;
    Else
        Return True;
    End If
End Function

' use VBA keyword WithEvents to connect to object-level event
Dim WithEvents objView As XMLSpyLib.AuthenticView
Set objView = objSpy.ActiveDocument.AuthenticView

' continue here with something useful ...
' and serve the windows message loop
```

OnBeforePaste

See also

Event: *OnBeforePaste* (*objData* as Variant, *strType* as String) as Boolean

XMLSpy scripting environment - VBScript:

```
Function On_AuthenticBeforePaste(objData, strType)
    'On_AuthenticBeforePaste=False 'to disable operation
EndFunction
```

XMLSpy scripting environment - JScript:

```
function On_AuthenticBeforePaste(objData, strType)
{
    //return False; 'to disable operation
}
```


XMLSpy IDE Plugin:

```
IXMLSpyPlugIn.OnEvent (19, ...) //nEventId=19
```

Description

This event gets triggered before a paste operation gets performed on the document. The parameter *strType* is one of "TEXT", "UNICODETEXT" or "IUNKNOWN". In the first two cases *objData* contains a string representation of the object that will be pasted. In the later case, *objData* contains a pointer to an IUnknown COM interface.

Return *True* (or nothing) to allow paste operation. Return *False* to disable operation.

OnBeforeSave

Event: *OnBeforeSave* (SaveAs flag) as Boolean

Description: *OnBeforeSave* gives the opportunity to e.g. warn the user about overwriting the existing XML document, or to make the document read-only when specific circumstances are not met. The event will be fired before the file dialog is shown. (Please note, that the event fires when saving the XML document, and not when saving the SPS design in StyleVision.)

OnDragOver

See also

Event: *OnDragOver* (*nXPos* as Long, *nYPos* as Long, *eMouseEvent* as [SPYMouseEvent](#), *objRange* as [AuthenticRange](#), *objData* as [AuthenticDataTransfer](#)) as Boolean

XMLSpy scripting environment - VBScript:

```
Function On_AuthenticDragOver(nXPos, nYPos, eMouseEvent, objRange,  
    objData)  
    'On_AuthenticDragOver=False 'to disable operation  
EndFunction
```

XMLSpy scripting environment - JScript:

```
function On_AuthenticDragOver(nXPos, nYPos, eMouseEvent, objRange, objData)  
{  
    //On_AuthenticDragOver=False 'to disable operation  
}
```

XMLSpy IDE Plugin:

```
IXMLSpyPlugIn.OnEvent (10, ...) //nEventId=10
```

Description

This event gets triggered whenever an object from within our outside of Authentic View gets dragged with the mouse over the application window. All event related information gets passed as parameters.

The first three parameters specify the mouse position, the mouse button status and the status of the virtual keys at the time when the event occurred. The parameter *objRange* passes a range object that selects the XML element below the mouse position. The value of this parameter might be *NULL*. Be sure to check before you access the range object. The parameter *objData* allows to access information about the object being dragged.

Return *False* to cancel the drag operation. Return *True* (or nothing) to continue normal

operation.

Examples

```
'
'-----
' VB code snippet - connecting to object level events
'-----
' access XMLSpy (without checking for any errors)
Dim objSpy As XMLSpyLib.Application
Set objSpy = GetObject("", "XMLSpy.Application")

' this is the event callback routine connected to the OnDragOver
' event of object objView
Private Function objView_OnDragOver(ByVal i_nXPos As Long, ByVal i_nYPos As
Long,
                                ByVal i_eMouseEvent As SPYMouseEvent,
                                ByVal i_ipRange As IAuthenticRange,
                                ByVal i_ipData As
IAuthenticDataTransfer) As Boolean

    If (((i_eMouseEvent And spyShiftKeyDownMask) <> 0) And
        (Not i_ipRange Is Nothing)) Then
        MsgBox ("Floating over element " &
i_ipRange.FirstXMLData.Parent.Name);
    End If

    Return True;
End Function

' use VBA keyword WithEvents to connect to object-level event
Dim WithEvents objView As XMLSpyLib.AuthenticView
Set objView = objSpy.ActiveDocument.AuthenticView

' continue here with something useful ...
' and serve the windows message loop
```

OnKeyboardEvent

See also

Event: `OnKeyboardEvent` (`eKeyEvent` as [SPYKeyEvent](#), `nKeyCode` as Long, `nVirtualKeyStatus` as Long) as Boolean

XMLSpy scripting environment - VBScript:

```
Function On_AuthenticKeyboardEvent(eKeyEvent, nKeyCode,
nVirtualKeyStatus)
    'On_AuthenticKeyboardEvent=True 'to cancel bubbling of event
EndFunction
```

XMLSpy scripting environment - JScript:

```
function On_AuthenticKeyboardEvent(eKeyEvent, nKeyCode, nVirtualKeyStatus)
{
    //return true; //to cancel bubbling of event
}
```

XMLSpy IDE Plugin:

```
IXMLSpyPlugin.OnEvent (30, ...) //nEventId=30
```

Description

This event gets triggered for `WM_KEYDOWN`, `WM_KEYUP` and `WM_CHAR` Windows

messages.

The actual message type is available in the *eKeyEvent* parameter. The status of virtual keys is combined in the parameter *nVirtualKeyStatus*. Use the bit-masks defined in the enumeration datatype [SPYVirtualKeyMask](#), to test for the different keys or their combinations.

OnLoad

Event: OnLoad ()

Description: OnLoad can be used e.g. to restrict some AuthenticView functionality, as shown in the example below:

```
function On_AuthenticLoad( )
{
    // We are disabling all entry helpers in order to prevent user from
    // manipulating XML tree
    AuthenticView.DisableElementEntryHelper();
    AuthenticView.DisableAttributeEntryHelper();

    // We are also disabling the markup buttons for the same purpose
    AuthenticView.SetToolBarButtonState( 'AuthenticMarkupSmall',
authenticToolBarButtonDisabled );
    AuthenticView.SetToolBarButtonState( 'AuthenticMarkupLarge',
authenticToolBarButtonDisabled );
    AuthenticView.SetToolBarButtonState( 'AuthenticMarkupMixed',
authenticToolBarButtonDisabled );
}
```

In the example the status of the Markup Small, Markup Large, Markup Mixed toolbar buttons are manipulated with the help of button identifiers. See [complete list](#).

OnMouseEvent

See also

Event: OnMouseEvent (*nXPos* as Long, *nYPos* as Long, *eMouseEvent* as [SPYMouseEvent](#), *objRange* as [AuthenticRange](#)) as Boolean

XMLSpy scripting environment - VBScript:

```
Function On_AuthenticMouseEvent(nXPos, nYPos, eMouseEvent, objRange)
    'On_AuthenticMouseEvent=True 'to cancel bubbling of event
EndFunction
```

XMLSpy scripting environment - JScript:

```
function On_AuthenticMouseEvent(nXPos, nYPos, eMouseEvent, objRange)
{
    //return true; //to cancel bubbling of event
}
```

XMLSpy IDE Plugin:

```
IXMLSpyPlugIn.OnEvent (31, ...) //nEventId=31
```

Description

This event gets triggered for every mouse movement and mouse button Windows message.

The actual message type and the mouse buttons status, is available in the *eMouseEvent* parameter. Use the bit-masks defined in the enumeration datatype [SPYMouseEvent](#) to test for

the different messages, button status, and their combinations.

The parameter *objRange* identifies the part of the document found at the current mouse cursor position. The range objects always selects a complete tag of the document. (This might change in future versions, when a more precise positioning mechanism becomes available). If no selectable part of the document is found at the current position, the range object is *null*.

OnSelectionChanged

See also

Event: *OnSelectionChanged* (*objNewSelection* as [AuthenticRange](#))

XMLSpy scripting environment - VBScript:

```
Function On_AuthenticSelectionChanged(objNewSelection)
EndFunction
```

XMLSpy scripting environment - JScript:

```
function On_AuthenticSelectionChanged(objNewSelection)
{
}
}
```

XMLSpy IDE Plugin:

```
IXMLSpyPlugin.OnEvent (23, ...) //nEventId=23
```

Description

This event gets triggered whenever the selection in the user interface changes.

Examples

```
'
'-----
' VB code snippet - connecting to object level events
'-----
' access XMLSpy (without checking for any errors)
Dim objSpy As XMLSpyLib.Application
Set objSpy = GetObject("", "XMLSpy.Application")

' this is the event callback routine connected to the OnSelectionChanged
' event of object objView
Private Sub objView_OnSelectionChanged (ByVal i_ipNewRange As
XMLSpyLib.IAuthenticRange)
    MsgBox ("new selection: " & i_ipNewRange.Text)
End Sub

' use VBA keyword WithEvents to connect to object-level event
Dim WithEvents objView As XMLSpyLib.AuthenticView
Set objView = objSpy.ActiveDocument.AuthenticView

' continue here with something useful ...
' and serve the windows message loop
```

OnToolbarButtonClicked

Event: *OnToolbarButtonClicked* (Button identifier)

Description: *OnToolbarButtonClicked* is fired when a toolbar button was clicked by user. The parameter button identifier helps to determine which button was clicked. The list of predefined button identifiers is below:

- AuthenticPrint
- AuthenticPrintPreview
- AuthenticUndo
- AuthenticRedo
- AuthenticCut
- AuthenticCopy
- AuthenticPaste
- AuthenticClear
- AuthenticMarkupHide
- AuthenticMarkupLarge
- AuthenticMarkupMixed
- AuthenticMarkupSmall
- AuthenticValidate
- AuthenticChangeWorkingDBXMLCell
- AuthenticSave
- AuthenticSaveAs
- AuthenticReload
- AuthenticTableInsertRow
- AuthenticTableAppendRow
- AuthenticTableDeleteRow
- AuthenticTableInsertCol
- AuthenticTableAppendCol
- AuthenticTableDeleteCol
- AuthenticTableJoinCellRight
- AuthenticTableJoinCellLeft
- AuthenticTableJoinCellAbove
- AuthenticTableJoinCellBelow
- AuthenticTableSplitCellHorizontally
- AuthenticTableSplitCellVertically
- AuthenticTableAlignCellContentTop
- AuthenticTableCenterCellVertically
- AuthenticTableAlignCellContentBottom
- AuthenticTableAlignCellContentLeft
- AuthenticTableCenterCellContent
- AuthenticTableAlignCellContentRight
- AuthenticTableJustifyCellContent
- AuthenticTableInsertTable
- AuthenticTableDeleteTable
- AuthenticTableProperties
- AuthenticAppendRow
- AuthenticInsertRow
- AuthenticDuplicateRow
- AuthenticMoveRowUp
- AuthenticMoveRowDown
- AuthenticDeleteRow
- AuthenticDefineEntities
- AuthenticXMLSignature

For custom buttons the user might add his own identifiers. Please, note that the user must take

care, as the identifiers are not checked for uniqueness. The same identifiers can be used to identify buttons in the `Set/GetToolBarState()` COM API calls. By adding code for different buttons, the user is in the position to completely redefine the AuthenticView toolbar behavior, adding own methods for table manipulation, etc.

OnToolBarButtonExecuted

Event: `OnToolBarButtonExecuted` (Button identifier)

Description: `OnToolBarButtonClicked` is fired when a toolbar button was clicked by user. The parameter button identifier helps to determine which button was clicked. See the list of [predefined button identifiers](#).

`OnToolBarButtonExecuted` is fired after the toolbar action was executed. It is useful e.g. to add update code, as shown in the example below:

```
//event fired when a toolbar button action was executed
function On_AuthenticToolBarButtonExecuted( varBtnIdentifier )
{
    // After whatever command user has executed - make sure to update
    toolbar button states
    UpdateOwnToolBarButtonStates();
}
```

In this case `UpdateOwnToolBarButtonStates` is a user function defined in the Global Declarations.

OnUserAddedXMLNode

Event: `OnUserAddedXMLNode` (XML node)

Description: `OnUserAddedXMLNode` will be fired when the user adds an XML node as a primary action. This happens in the situations, where the user clicks on

- auto-add hyperlinks (see example `OnUserAddedXMLNode.sps`)
- the Insert..., Insert After..., Insert Before... context menu items
- Append row, Insert row toolbar buttons
- Insert After..., Insert Before... actions in element entry helper (outside StyleVision)

The event doesn't get fired on Duplicate row, or when the node was added externally (e.g. via COM API), or on Apply (e.g. Text State Icons), or when in XML table operations or in DB operations.

The event parameter is the XML node object, which was added giving the user an opportunity to manipulate the XML node added. An elaborate example for an event handler can be found in the `OnUserAddedXMLNode.sps` file.

Application

See also

Property: `Application` as [Application](#) (read-only)

Description

Accesses the XMLSpy application object.

Errors

- 2000 The authentic view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.

AsXMLString**See also**

Property: [AsXMLString](#) as String

Description

Returns or sets the document content as an XML string. Setting the content to a new value does not change the schema file or sps file in use. If the new `XMLString` does not match the actual schema file error 2011 gets returned.

Errors

- 2000 The authentic view object is no longer valid.
- 2011 `AsXMLString` was set to a value which is no valid XML for the current schema file.

ContextMenu

Property: [ContextMenu\(\)](#) as `ContextMenu`

Description

The property `ContextMenu` gives access to customize the context menu. The best place to do it is in the event handler `OnContextMenuActivated`.

Errors

- 2000 Invalid object.
- 2005 Invalid parameter.

CreateXMLNode

Method: [CreateXMLNode](#) (*nKind* as [SPYXMLDataKind](#)) as [XMLData](#)

Return Value

The method returns the new [XMLData](#) object.

Description

To create a new `XMLData` object use the `CreateXMLNode()` method. See also [Using XMLData](#).

Errors

- 2000 Invalid object.
- 2012 Cannot create XML node.

DisableAttributeEntryHelper

Method: [DisableAttributeEntryHelper\(\)](#)

Description

`DisableAttributeEntryHelper()` disables the attribute entry helper in XMLSpy, Authentic Desktop and Authentic Browser plug-in.

Errors

- 2000 Invalid object.

DisableElementEntryHelper

Method: `DisableElementEntryHelper()`

Description

`DisableElementEntryHelper()` disables the element entry helper in XMLSpy, Authentic Desktop and Authentic Browser plug-in.

Errors

2000 Invalid object.

DisableEntityEntryHelper

Method: `DisableEntityEntryHelper()`

Description

`DisableEntityEntryHelper()` disables the entity entry helper in XMLSpy, Authentic Desktop and Authentic Browser plug-in.

Errors

2000 Invalid object.

DocumentBegin

See also

Property: `DocumentBegin` as [AuthenticRange](#) (read-only)

Description

Retrieve a range object that points to the beginning of the document.

Errors

2000 The authentic view object is no longer valid.
2005 Invalid address for the return parameter was specified.

DocumentEnd

See also

Property: `DocumentEnd` as [AuthenticRange](#) (read-only)

Description

Retrieve a range object that points to the end of the document.

Errors

2000 The authentic view object is no longer valid.
2005 Invalid address for the return parameter was specified.

DoNotPerformStandardAction

Method: `DoNotPerformStandardAction()`

Description

`DoNotPerformStandardAction()` serves as cancel bubble for macros, and stops further execution after macro has finished.

Errors

2000 Invalid object.

EvaluateXPath

Method: `EvaluateXPath` (XMLData as [XMLData](#), strExpression as string) strValue as string

Return Value

The method returns a string

Description

`EvaluateXPath()` executes an XPath expressions with the given XML context node. The result is returned as string, in the case of a sequence it is a space-separated string.

Errors

2000 Invalid object.
2005 Invalid parameter.
2008 Internal error.
2013 XPath error.

Event

See also

Property: `Event` as [AuthenticEvent](#) (read-only)

Description

This property gives access to parameters of the last event in the same way as [OidAuthenticView.event](#) does. Since all events for the scripting environment and external clients are now available with parameters this `Event` property should only be used from within IDE-Plugins.

Errors

2000 The authentic view object is no longer valid.
2005 Invalid address for the return parameter was specified.

EventContext

Property: `EventContext()` as [EventContext](#)

Description

`EventContext` property gives access to the running macros context. See the [EventContext](#) interface description for more details.

Errors

2000 Invalid object.

GetToolbarButtonState

Method: `GetToolbarButtonState` (ButtonIdentifier as string) as AuthenticToolbarButtonState

Return Value

The method returns `AuthenticToolBarButtonState`

Description

`Get/SetToolBarButtonState` queries the status of a toolbar button, and lets the user disable or enable the button, identified via its button identifier ([see list above](#)). One usage is to disable toolbar buttons permanently. Another usage is to put `SetToolBarButtonState` in the `OnSelectionChanged` event handler, as toolbar buttons are updated regularly when the selection changes in the document.

Toolbar button states are given by the [listed enumerations](#).

The default state means that the enable/disable of the button is governed by `AuthenticView`. When the user sets the button state to enable or disable, the button remains in that state as long as the user does not change it.

Errors

- 2000 Invalid object.
- 2005 Invalid parameter.
- 2008 Internal error.
- 2014 Invalid button identifier.

Goto**See also**

Method: `Goto` (*eKind* as [SPYAuthenticElementKind](#), *nCount* as Long, *eFrom* as [SPYAuthenticDocumentPosition](#)) as [AuthenticRange](#)

Description

Retrieve a range object that points to the beginning of the *nCount* element of type *eKind*. The start position is defined by the parameter *eFrom*. Use positive values for *nCount* to navigate to the document end. Use negative values to navigate towards the beginning of the document.

Errors

- 2000 The authentic view object is no longer valid.
- 2003 Target lies after end of document.
- 2004 Target lies before beginning of document.
- 2005 Invalid element kind specified.
The document position to start from is not one of `spyAuthenticDocumentBegin` or `spyAuthenticDocumentEnd`.
Invalid address for the return parameter was specified.

Examples

```
' -----
' XMLSpy scripting environment - VBScript
' -----

Dim objAuthenticView
' we assume that the active document is open in authentic view mode
Set objAuthenticView = Application.ActiveDocument.AuthenticView

On Error Resume Next
Dim objRange
' goto beginning of first table in document
Set objRange = objAuthenticView.Goto (spyAuthenticTable, 1,
spyAuthenticDocumentBegin)
If (Err.number = 0) Then
```

```
        objRange.Select()  
    Else  
        MsgBox "No table found in document"  
    End If
```

IsRedoEnabled

See also

Property: [IsRedoEnabled](#) as Boolean (read-only)

Description

True if redo steps are available and [Redo](#) is possible.

Errors

- 2000 The authentic view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.

IsUndoEnabled

See also

Property: [IsUndoEnabled](#) as Boolean (read-only)

Description

True if undo steps are available and [Undo](#) is possible.

Errors

- 2000 The authentic view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.

MarkupVisibility

See also

Property: [MarkupVisibility](#) as [SPYAuthenticMarkupVisibility](#)

Description

Set or get current visibility of markup.

Errors

- 2000 The authentic view object is no longer valid.
- 2005 Invalid enumeration value was specified.
Invalid address for the return parameter was specified.

Parent

See also

Property: [Parent](#) as [Document](#) (read-only)

Description

Access the document shown in this view.

Errors

- 2000 The authentic view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.

Print**See also**

Method: `Print` (*bWithPreview* as Boolean, *bPromptUser* as Boolean)

Description

Print the document shown in this view. If *bWithPreview* is set to *True*, the print preview dialog pops up. If *bPromptUser* is set to *True*, the print dialog pops up. If both parameters are set to *False*, the document gets printed without further user interaction.

Errors

- 2000 The authentic view object is no longer valid.

Redo**See also**

Method: `Redo()` as Boolean

Description

Redo the modification undone by the last undo command.

Errors

- 2000 The authentic view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.

Selection**See also**

Property: `Selection` as [AuthenticRange](#)

Description

Set or get current text selection in user interface.

Errors

- 2000 The authentic view object is no longer valid.
- 2002 No cursor selection is active.
- 2005 Invalid address for the return parameter was specified.

Examples

```
' -----  
' XMLSpy scripting environment - VBScript  
' -----  
  
Dim objAuthenticView  
' we assume that the active document is open in authentic view mode  
Set objAuthenticView = Application.ActiveDocument.AuthenticView  
  
' if we are the end of the document, re-start at the beginning
```

```
If ( objAuthenticView.Selection.IsEqual( objAuthenticView.DocumentEnd) ) Then
    objAuthenticView.Selection = objAuthenticView.DocumentBegin
Else
    ' objAuthenticView.Selection =
objAuthenticView.Selection.GotoNextCursorPosition()
    ' or shorter:
    objAuthenticView.Selection.GotoNextCursorPosition().Select
End If
```

SetToolBarButtonState

Method: `SetToolBarButtonState` (ButtonIdentifier as string,
AuthenticToolBarButtonState state)

Description

`Get/SetToolBarButtonState` queries the status of a toolbar button, and lets the user disable or enable the button, identified via its button identifier ([see list above](#)). One usage is to disable toolbar buttons permanently. Another usage is to put `SetToolBarButtonState` in the `OnSelectionChanged` event handler, as toolbar buttons are updated regularly when the selection changes in the document.

Toolbar button states are given by the [listed enumerations](#).

The default state means that the enable/disable of the button is governed by `AuthenticView`. When the user sets the button state to enable or disable, the button remains in that state as long as the user does not change it.

Errors

- 2000 Invalid object.
- 2008 Internal error.
- 2014 Invalid button identifier.

Undo

See also

Method: `Undo()` as Boolean

Description

Undo the last modification of the document from within this view.

Errors

- 2000 The authentic view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.

UpdateXMLInstanceEntities

See also

Method: `UpdateXMLInstanceEntities()`

Description

Updates the internal representation of the declared entities, and refills the entry helper. In addition, the validator is reloaded, allowing the XML file to validate correctly. Please note that this may also cause schema files to be reloaded.

Errors

The method never returns an error.

Example

```
// -----
// XMLSpy scripting environment - JavaScript
// -----
if( Application.ActiveDocument &&
( Application.ActiveDocument.CurrentViewMode == 4) )
{
    var objDocType;
    objDocType =
Application.ActiveDocument.DocEditView.XMLRoot.GetFirstChild(10);

    if( objDocType)
    {
        var objEntity = Application.ActiveDocument.CreateChild(14);
        objEntity.Name = "child";
        objEntity.TextValue = "SYSTEM \"child.xml\"";
        objDocType.AppendChild( objEntity );

Application.ActiveDocument.AuthenticView.UpdateXMLInstanceEntities();
    }
}
```

WholeDocument**See also**

Property: [WholeDocument](#) as [AuthenticRange](#) (read-only)

Description

Retrieve a range object that selects the whole document.

Errors

- 2000 The authentic view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.

XMLDataRoot**See also**

Property: [XMLDataRoot](#) as [XMLData](#) (read-only)

Description

Returns or sets the top-level XMLData element of the current document. This element typically describes the document structure and would be of kind spyXMLDataXMLDocStruct, spyXMLDataXMLEntityDocStruct or spyXMLDataDTDDocStruct..

Errors

- 2000 The authentic view object is no longer valid.
- 2005 Invalid address for the return parameter was specified.

3.2.7 CodeGeneratorDlg**See also**

Only available/enabled in the Enterprise edition. An error is returned, if accessed by any other version.

Properties and Methods

Standard automation properties

[Application](#)

[Parent](#)

Programming language selection properties

[ProgrammingLanguage](#)

[TemplateName](#)

[CompatibilityMode](#)

Settings for C++ code

[CPPSettings_DOMType](#)

[CPPSettings_LibraryType](#)

[CPPSettings_UseMFC](#)

[CPPSettings_GenerateVC6ProjectFile](#)

[CPPSettings_GenerateVSPProjectFile](#)

Settings for C# code

[CSharpSettings_ProjectType](#)

Dialog handling for above code generation properties

[PropertySheetDlgAction](#)

Output path selection properties

[OutputPath](#)

[OutputPathDlgAction](#)

Presentation of result

[OutputResultDlgAction](#)

Description

Use this object to configure the generation of program code for schema files. The method [GenerateProgramCode](#) expects a `CodeGeneratorDlg` as parameter to configure code generation as well as the associated user interactions.

Application

See also

Only available/enabled in the Enterprise edition. An error is returned, if accessed by any other version.

Property: [Application](#) as [Application](#) (read-only)

Description

Access the XMLSpy application object.

Errors

- 2200 The object is no longer valid.
- 2201 Invalid address for the return parameter was specified.

CompatibilityMode

Property: [CompatibilityMode](#) as Boolean

Only available/enabled in the Enterprise edition. An error is returned, if accessed by any other version.

Description

Set to `true` to generate code compatible to XMLSpy 2005R3. Set to `false` to use newly added code-generation features.

Errors

- 2200 The object is no longer valid.
- 2201 Invalid action passed as parameter or an invalid address was specified for the return parameter.

CPPSettings_DOMType

Property: [CPPSettings_DOMType](#) as [SPYDOMType](#)

Only available/enabled in the Enterprise edition. An error is returned, if accessed by any other version.

Description

Defines one of the settings that configure generation of C++ code.

Errors

- 2200 The object is no longer valid.
- 2201 Invalid action passed as parameter or an invalid address was specified for the return parameter.

CPPSettings_GenerateVC6ProjectFile

Property: [CPPSettings_GenerateVC6ProjectFile](#) as Boolean

Only available/enabled in the Enterprise edition. An error is returned, if accessed by any other version.

Description

Defines one of the settings that configure generation of C++ code.

Errors

- 2200 The object is no longer valid.
- 2201 Invalid action passed as parameter or an invalid address was specified for the return parameter.

CPPSettings_GenerateGCCMakefile

Property: [CPPSettings_GenerateGCCMakefile](#) as Boolean

Only available/enabled in the Enterprise edition. An error is returned, if accessed by any other version.

Description

Creates makefiles to compile the generated code under Linux with GCC.

Errors

- 2200 The object is no longer valid.
- 2201 Invalid action passed as parameter or an invalid address was specified for the return parameter.

CPPSettings_GenerateVSProjectFile

Property: [CSharpSettings_GenerateVSProjectFile](#) as [SPYProjectType](#)

Only available/enabled in the Enterprise edition. An error is returned, if accessed by any other version.

Description

Defines one of the settings that configure generation of C++ code. Only `spyVisualStudio2005Project (=4)` and `spyVisualStudio2008Project (=5)` and `spyVisualStudio2010Project (=6)` are valid project types.

Errors

- 2200 The object is no longer valid.
- 2201 Invalid action passed as parameter or an invalid address was specified for the return parameter.

CPPSettings_LibraryType

Property: [CPPSettings_LibraryType](#) as [SPYLibType](#)

Only available/enabled in the Enterprise edition. An error is returned, if accessed by any other version.

Description

Defines one of the settings that configure generation of C++ code.

Errors

- 2200 The object is no longer valid.
- 2201 Invalid action passed as parameter or an invalid address was specified for the return parameter.

CPPSettings_UseMFC

Property: [CPPSettings_UseMFC](#) as Boolean

Only available/enabled in the Enterprise edition. An error is returned, if accessed by any other version.

Description

Defines one of the settings that configure generation of C++ code.

Errors

- 2200 The object is no longer valid.
- 2201 Invalid action passed as parameter or an invalid address was specified for the return parameter.

CSharpSettings_ProjectType

Property: [CSharpSettings_ProjectType](#) as [SPYProjectType](#)

Only available/enabled in the Enterprise edition. An error is returned, if accessed by any other version.

Description

Defines the only setting to configure generation of C# code.

Errors

- 2200 The object is no longer valid.
- 2201 Invalid action passed as parameter or an invalid address was specified for the return parameter.

OutputPath

Property: [OutputPath](#) as String

Only available/enabled in the Enterprise edition. An error is returned, if accessed by any other version.

Description

Selects the base directory for all generated code.

Errors

- 2200 The object is no longer valid.
- 2201 Invalid address for the return parameter was specified.

OutputPathDialogAction

Property: [OutputPathDialogAction](#) as [SPYDialogAction](#)

Only available/enabled in the Enterprise edition. An error is returned, if accessed by any other version.

Description

Defines how the sub-dialog for selecting the code generation output path gets handled. Set this value to *spyDialogUserInput(2)* to show the dialog with the current value of the [OutputPath](#) property as default. Use *spyDialogOK(0)* to hide the dialog from the user.

Errors

- 2200 The object is no longer valid.
- 2201 Invalid action passed as parameter or an invalid address was specified for the return parameter.

OutputResultDialogAction

Property: [OutputResultDialogAction](#) as [SPYDialogAction](#)

Only available/enabled in the Enterprise edition. An error is returned, if accessed by any other version.

Description

Defines how the sub-dialog that asks to show the result of the code generation process gets handled. Set this value to *spyDialogUserInput(2)* to show the dialog. Use *spyDialogOK(0)* to hide the dialog from the user.

Errors

- 2200 The object is no longer valid.
- 2201 Invalid action passed as parameter or an invalid address was specified for the return parameter.

Parent

See also

Only available/enabled in the Enterprise edition. An error is returned, if accessed by any other version.

Property: [Parent](#) as [Dialogs](#) (read-only)

Description

Access the parent of the object.

Errors

- 2200 The object is no longer valid.
- 2201 Invalid address for the return parameter was specified.

ProgrammingLanguage

Property: [ProgrammingLanguage](#) as [ProgrammingLanguage](#)

Only available/enabled in the Enterprise edition. An error is returned, if accessed by any other version.

Description

Selects the output language for the code to be generated.

CAUTION: Setting this property to one of C++, C# or Java, changes the property [TemplateFileName](#) to the appropriate template file delivered with XMLSpy as well. If you want to generate C++, C# or Java code based on your own templates, set first the programming language and then select your template file.

Errors

- 2200 The object is no longer valid.
- 2201 Invalid address for the return parameter was specified.

PropertySheetDialogAction

Property: [PropertySheetDialogAction](#) as [SPYDialogAction](#)

Only available/enabled in the Enterprise edition. An error is returned, if accessed by any other version.

Description

Defines how the sub-dialog that configures the code generation process gets handled. Set this value to `spyDialogUserInput(2)` to show the dialog with the current values as defaults. Use `spyDialogOK(0)` to hide the dialog from the user.

Errors

- 2200 The object is no longer valid.
- 2201 Invalid action passed as parameter or an invalid address was specified for the return parameter.

TemplateFileName

Property: [TemplateFileName](#) as String

Only available/enabled in the Enterprise edition. An error is returned, if accessed by any other version.

Description

Selects the code generation template file. XMLSpy comes with template files for C++, C# or Java in the SPL folder of your installation directory.

Setting this property to one of the code generation template files of your XMLSpy installation automatically sets the [ProgrammingLanguage](#) property to its appropriate value.

Errors

- 2200 The object is no longer valid.
- 2201 Invalid address for the return parameter was specified.

3.2.8 DatabaseConnection

See also

Properties for import and export

[File](#) or
[ADOConnection](#) or
[ODBCConnection](#)

Properties for import only

[DatabaseKind](#)
[SQLSelect](#)
[AsAttributes](#)
[ExcludeKeys](#)
[IncludeEmptyElements](#)
[NumberDateTimeFormat](#)
[NullReplacement](#)
[CommentIncluded](#)

Properties for export only

[CreateMissingTables](#)
[CreateNew](#)
[TextFieldLen](#)
[DatabaseSchema](#)

Properties for XML Schema from DB Structure generation

[PrimaryKeys](#)
[ForeignKeys](#)
[UniqueKeys](#)
[SchemaExtensionType](#)
[SchemaFormat](#)
[ImportColumnsType](#)

Description

`DatabaseConnection` specifies the parameters for the database connection.

Please note that the properties of the `DatabaseConnection` interface are referring to the settings of the import and export dialogs of XMLSpy.

ADOConnection

See also

Property: [ADOConnection](#) as [String](#)

Description

The property `ADOConnection` contains a connection string. Either use this property or [ODBCConnection](#) or [File](#) to refer to a database.

Errors

No error codes are returned.

Example

```
Dim objSpyConn As DatabaseConnection
Set objSpyConn = objSpy.GetDatabaseSettings

Dim objADO As DataLinks
Set objADO = CreateObject("DataLinks")

If Not (objADO Is Nothing) Then
    Dim objConn As Connection
    Set objConn = objADO.PromptNew
    objSpyConn.ADOConnection = objConn.ConnectionString
End If
```

AsAttributes

See also

Property: [AsAttributes](#) as [Boolean](#)

Description

Set `AsAttributes` to true if you want to initialize all import fields to be imported as attributes. Default is false and will initialize all fields to be imported as elements. This property is used only in calls to [Application.GetDatabaseImportElementList](#).

Errors

No error codes are returned.

CommentIncluded

See also

Property: [CommentIncluded](#) as [Boolean](#)

Description

This property tells whether additional comments are added to the generated XML. Default is true. This property is used only when importing from databases.

Errors

No error codes are returned.

CreateMissingTables

See also

Property: [CreateMissingTables](#) as Boolean

Description

If `CreateMissingTables` is true, tables which are not already defined in the export database will be created during export. Default is true. This property is used only when exporting to databases.

Errors

No error codes are returned.

CreateNew

See also

Property: [CreateNew](#) as Boolean

Description

Set `CreateNew` true if you want to create a new database on export. Any existing database will be overwritten. See also [DatabaseConnection.File](#). Default is false. This property is used only when exporting to databases.

Errors

No error codes are returned.

DatabaseKind

See also

Property: [DatabaseKind](#) as [SPYDatabaseKind](#)

Description

Select the kind of database that gets access. The default value is `spyDB_Unspecified(7)` and is sufficient in most cases. This property is used only when importing from databases.

Errors

No error codes are returned.

DatabaseSchema

See also

Property: [DatabaseSchema](#) as String

Description

This property specifies the Schema used for export in Schema aware databases. Default is "". This property is used only when exporting to databases.

Errors

No error codes are returned.

ExcludeKeys

See also

Property: [ExcludeKeys](#) as Boolean

Description

Set [ExcludeKeys](#) to true if you want to exclude all key columns from the import data. Default is false. This property is used only when importing from databases.

Errors

No error codes are returned.

File

See also

Property: [File](#) as String

Description

The property [File](#) sets the path for the database during export or import. This property can only be used in conjunction with a Microsoft Access database. Either use this property or [ODBCConnection](#) or [ADOConnection](#) to refer to the database.

See also [Import and Export](#).

Errors

No error codes are returned.

ForeignKeys

See also

Property: [ForeignKeys](#) as Boolean

Description

Specifies whether the Foreign Keys constraint is created or not. Default is true. This property is used only when creating a XML Schema from a DB structure.

Errors

No error codes are returned.

ImportColumnsType

See also

Property: [ImportColumnsType](#) as [SPYImportColumnsType](#)

Description

Defines if column information from the DB is saved as element or attribute in the XML Schema. Default is as element. This property is used only when creating a XML Schema from a DB structure.

Errors

No error codes are returned.

IncludeEmptyElements

See also

Property: [IncludeEmptyElements](#) as Boolean

Description

Set `IncludeEmptyElements` to false if you want to exclude all empty elements. Default is true. This property is used only when importing from databases.

Errors

No error codes are returned.

NullReplacement

See also

Property: [NullReplacement](#) as String

Description

This property contains the text value that is used during import for empty elements (null values). Default is "". This property is used only when importing from databases.

Errors

No error codes are returned.

NumberDateTimeFormat

See also

Property: [NumberDateTimeFormat](#) as [SPYNumberDateTimeFormat](#)

Description

The property `NumberDateTimeFormat` sets the format of numbers and date- and time-values. Default is [spySystemLocale](#). This property is used only when importing from databases.

Errors

No error codes are returned.

ODBCConnection

See also

Property: [ODBCConnection](#) as String

Description

The property `ODBCConnection` contains a ODBC connection string. Either use this property or [ADOConnection](#) or [File](#) to refer to a database.

Errors

No error codes are returned.

PrimaryKeys

See also

Property: `PrimaryKeys` as `Boolean`

Description

Specifies whether the Primary Keys constraint is created or not. Default is true. This property is used only when creating a XML Schema from a DB structure.

Errors

No error codes are returned.

SchemaExtensionType

See also

Property: `SchemaExtensionType` as `SPYSchemaExtensionType`

Description

Defines the Schema extension type used during the Schema generation. This property is used only when creating a XML Schema from a DB structure.

See also [Create XML Schema from DB Structure](#).

Errors

No error codes are returned.

SchemaFormat

See also

Property: `SchemaFormat` as `SPYSchemaFormat`

Description

Defines the Schema format used during the Schema generation. This property is used only when creating a XML Schema from a DB structure.

See also [Create XML Schema from DB Structure](#).

Errors

No error codes are returned.

SQLSelect

See also

Property: `SQLSelect` as `String`

Description

The SQL query for the import is stored in the property `SQLSelect`. This property is used only when importing from databases. See also [Import and Export](#).

Errors

No error codes are returned.

TextFieldLen

See also

Property: [TextFieldLen](#) as `Int`

Description

The property `TextFieldLen` sets the length for created text fields during the export. Default is 255. This property is used only when exporting to databases.

Errors

No error codes are returned.

UniqueKeys

See also

Property: [UniqueKeys](#) as `Boolean`

Description

Specifies whether the Unique Keys constraint is created or not. Default is true. This property is used only when creating a XML Schema from a DB structure.

Errors

No error codes are returned.

3.2.9 Dialogs

See also

Properties and Methods

Standard automation properties

[Application](#)

[Parent](#)

Various dialog objects

[CodeGeneratorDlg](#)

[FileSelectionDlg](#)

[SchemaDocumentationDlg](#)

[GenerateSampleXMLDlg](#)

[DTDSchemaGeneratorDlg](#)

[FindInFilesDlg](#)

[WSDLDocumentationDlg](#)

[WSDL20DocumentationDlg](#)

[XBRLDocumentationDlg](#)

Description

The Dialogs object provides access to different built-in dialogs of XMLSpy. These dialog objects allow to initialize the fields of user dialogs before they get presented to the user or allow to simulate complete user input by your program.

Application

See also

Property: [Application](#) as [Application](#) (read-only)

Description

Access the XMLSpy application object.

Errors

- 2300 The object is no longer valid.
- 2301 Invalid address for the return parameter was specified.

CodeGeneratorDlg

See also

Only available/enabled in the Enterprise edition. An error is returned, if accessed by any other version.

Property: [CodeGeneratorDlg](#) as [CodeGeneratorDlg](#) (read-only)

Description

Get a new instance of a code generation dialog object. You will need this object to pass the necessary parameters to the code generation methods. Initial values are taken from last usage of the code generation dialog.

Errors

- 2300 The Dialogs object or one of its parents is no longer valid.
- 2301 Invalid address for the return parameter was specified.

FileSelectionDlg

See also

Property: [FileSelectionDlg](#) as [FileSelectionDlg](#) (read-only)

Description

Get a new instance of a file selection dialog object.

File selection dialog objects are passed to you with the some events that signal opening or saving of documents and projects.

Errors

- 2300 The Dialogs object or one of its parents is no longer valid.
- 2301 Invalid address for the return parameter was specified.

Parent

See also

Property: [Parent](#) as [Application](#) (read-only)

Description

Access the XMLSpy application object.

Errors

- 2300 The object is no longer valid.
- 2301 Invalid address for the return parameter was specified.

SchemaDocumentationDlg**See also**

Property: [SchemaDocumentationDlg](#) as [SchemaDocumentationDlg](#) (read-only)

Description

Get a new instance of a dialog object that parameterizes generation of schema documentation. See [Document.GenerateSchemaDocumentation](#) for its usage.

Errors

- 2300 The Dialogs object or one of its parents is no longer valid.
- 2301 Invalid address for the return parameter was specified.

GenerateSampleXMLDlg**See also**

Property: [GenerateSampleXMLDlg](#) as [GenerateSampleXMLDlg](#) (read-only)

Description

Get a new instance of a dialog object that parameterizes generation of a sample XML based on a W3C schema or DTD. See [GenerateSampleXML](#) for its usage.

Errors

- 2300 The Dialogs object or one of its parents is no longer valid.
- 2301 Invalid address for the return parameter was specified.

DTDSchemaGeneratorDlg**See also**

Property: [DTDSchemaGeneratorDlg](#) as [DTDSchemaGeneratorDlg](#) (read-only)

Description

Get a new instance of a dialog object that parameterizes generation of a schema or DTD. See [Document.GenerateDTDOrSchemaEx](#) for its usage.

Errors

- 2300 The Dialogs object or one of its parents is no longer valid.
- 2301 Invalid address for the return parameter was specified.

FindInFilesDlg**See also**

Property: [FindInFilesDlg](#) as [FindInFilesDlg](#) (read-only)

Description

Get a new instance of a dialog object that parameterizes the search (or replacement) of strings in files. See [Application.FindInFiles](#) for its usage.

Errors

- 2300 The Dialogs object or one of its parents is no longer valid.
- 2301 Invalid address for the return parameter was specified.

WSDLDocumentationDlg**See also**

Property: [WSDLDocumentationDlg](#) as [WSDLDocumentationDlg](#) (read-only)

Description

Get a new instance of a dialog object that parameterizes generation of WSDL documentation. See [Document.GenerateWSDLDocumentation](#) for its usage.

Errors

- 2300 The Dialogs object or one of its parents is no longer valid.
- 2301 Invalid address for the return parameter was specified.

WSDL20DocumentationDlg**See also**

Property: [WSDL20DocumentationDlg](#) as [WSDL20DocumentationDlg](#) (read-only)

Description

Get a new instance of a dialog object that parameterizes generation of WSDL 2.0 documentation. See [Document.GenerateWSD20LDocumentation](#) for its usage.

Errors

- 2300 The Dialogs object or one of its parents is no longer valid.
- 2301 Invalid address for the return parameter was specified.

XBRLDocumentationDlg**See also**

Property: [XBRL20DocumentationDlg](#) as [XBRL20DocumentationDlg](#) (read-only)

Description

Get a new instance of a dialog object that parameterizes generation of WSDL 2.0 documentation. See [Document.GenerateXBRLDocumentation](#) for its usage.

Errors

- 2300 The Dialogs object or one of its parents is no longer valid.
- 2301 Invalid address for the return parameter was specified.

3.2.10 Document**See also****Properties and Methods**

Standard automation properties
[Application](#)

[Parent](#)

Various document properties and methods

[SetActiveDocument](#)[Encoding](#)[SetEncoding \(obsolete\)](#)[Suggestions](#)

XML validation

[IsWellFormed](#)[IsValid](#)[SetExternalIsValid](#)

Document conversion and transformation

[AssignDTD](#)[AssignSchema](#)[AssignXSL](#)[AssignXSLFO](#)[ConvertDTDOrSchema](#)[ConvertDTDOrSchemaEx](#)[GenerateDTDOrSchema](#)[GenerateDTDOrSchemaEx](#)[CreateSchemaDiagram](#)[ExecuteXQuery](#)[TransformXSL](#)[TransformXSLEx](#)[TransformXSLFO](#)[GenerateProgramCode](#) (Enterprise Edition only)[GenerateSchemaDocumentation](#)[GenerateSampleXML](#)[GenerateWSDL20Documentation](#)[GenerateWSDLDocumentation](#)[GenerateXBRLDocumentation](#)[ConvertToWSDL20](#)

Document export

[GetExportElementList](#)[ExportToText](#)[ExportToDatabase](#)[CreateDBStructureFromXMLSchema](#)[GetDBStructureList](#)

File saving and naming

[FullName](#)[Name](#)[Path](#)[GetPathName \(obsolete\)](#)[SetPathName \(obsolete\)](#)[Title](#)[IsModified](#)[Saved](#)[SaveAs](#)[Save](#)[SaveInString](#)[SaveToURL](#)[Close](#)

View access

[CurrentViewMode](#)

[SwitchViewMode](#)
[AuthenticView](#)
[GridView](#)
[DocEditView \(obsolete\)](#)

Access to XMLData

[RootElement](#)
[DataRoot](#)
[CreateChild](#)
[UpdateViews](#)
[StartChanges](#)
[EndChanges](#)
[UpdateXMLData](#)

Description

Document objects represent XML documents opened in XMLSpy.

Use one of the following properties to access documents that are already open XMLSpy:

[Application.ActiveDocument](#)
[Application.Documents](#)

Use one of the following methods to open a new document in XMLSpy:

[Documents.OpenFile](#)
[Documents.OpenURL](#)
[Documents.OpenURLDialog](#)
[Documents.NewFile](#)
[Documents.NewFileFromText](#)
[SpyProjectItem.Open](#)
[Application.ImportFromDatabase](#)
[Application.ImportFromSchema](#)
[Application.ImportFromText](#)
[Application.ImportFromWord](#)
[Document.ConvertDTDOrSchema](#)
[Document.GenerateDTDOrSchema](#)

Events

OnBeforeSaveDocument

See also

Event: `OnBeforeSaveDocument(objDocument as Document, objDialog as FileSelectionDlg)`

XMLSpy scripting environment - VBScript:

```

Function On_BeforeSaveDocument(objDocument, objDialog)
EndFunction

' old handler - now obsolete
' return string to save to new file name
' return empty string to cancel save operation
' return nothing to save to original name
Function On_SaveDocument(objDocument, strFilePath)
EndFunction

```

XMLSpy scripting environment - JScript:

```

from On_BeforeSaveDocument(objDocument, objDialog)

```

```

{
}

// old handler - now obsolete
// return string to save to new file name
// return empty string to cancel save operation
// return nothing to save to original name
function On_SaveDocument(objDocument, strFilePath)
{
}

```

XMLSpy IDE Plugin:

```
IXMLSpyPlugin.OnEvent (27, ...) //nEventId=27
```

Description

This event gets fired on any attempt to save a document. The file selection dialog object is initialized with the name chosen for the document file. You can modify this selection. To continue saving the document leave the [FileSelectionDialogAction](#) property of *io_objDialog* at its default value [spyDialogOK](#). To abort saving of the document set this property to [spyDialogCancel](#).

OnBeforeCloseDocument**See also**

Event: *OnBeforeCloseDocument*(*objDocument* as [Document](#)) as Boolean

XMLSpy scripting environment - VBScript:

```

Function On_BeforeCloseDocument(objDocument)
    'On_BeforeCloseDocument=False 'to prohibit saving of document
EndFunction

```

XMLSpy scripting environment - JScript:

```

function On_BeforeCloseDocument(objDocument)
{
    //return false; 'to prohibit saving of document'
}

```

XMLSpy IDE Plugin:

```
IXMLSpyPlugin.OnEvent (28, ...) //nEventId=28
```

Description

This event gets fired on any attempt to close a document. To prevent the document from being closed return false.

OnBeforeValidate**See also**

Event: *OnBeforeValidate*(*objDocument* as [Document](#), *bOnLoading* as Boolean, *bOnCommand* as Boolean) as Boolean

XMLSpy scripting environment - VBScript:

```

Function On_BeforeValidate(objDocument, bOnLoading, bOnCommand)
    On_BeforeValidate=bCancelDefaultValidation 'set by the script if

```



```
necessary
EndFunction
```

XMLSpy scripting environment - JScript:

```
function On_BeforeValidate(objDocument, eViewMode, bActivated)
{
    return bCancelDefaultValidation //set by the script if necessary
}
```

XMLSpy IDE Plugin:

```
IXMLSpyPlugin.OnEvent (32, ...) //nEventId=32
```

Description

This event gets fired before the document is validated. It is possible to suppress the default validation by returning false from the event handler. In this case the script should also set the validation result using the [SetExternalsValid](#) method.

bOnLoading is true if the the event is raised on the initial validation on loading the document.

bOnCommand is true whenever the user selected the Validate command from the Toolbar or menu.

Available with TypeLibrary version 1.5

OnCloseDocument

See also

Event: OnCloseDocument(objDocument as [Document](#))

XMLSpy scripting environment - VBScript:

```
Function On_Close Document(objDocument)
EndFunction
```

XMLSpy scripting environment - JScript:

```
function On_Close Document(objDocument)
{
}
}
```

XMLSpy IDE Plugin:

```
IXMLSpyPlugin.OnEvent (8, ...) //nEventId=8
```

Description

This event gets fired as a result of closing a document. Do not modify the document from within this event.

OnViewActivation

See also

Event: OnViewActivation(objDocument as [Document](#), eViewMode as [SPYViewModes](#), bActivated as Boolean)

XMLSpy scripting environment - VBScript:

```
Function On_ViewActivation(objDocument, eViewMode, bActivated)
```

```
EndFunction
```

XMLSpy scripting environment - JScript:

```
function On_ViewActivation(objDocument, eViewMode, bActivated)
{
}
```

XMLSpy IDE Plugin:

```
IXMLSpyPlugin.OnEvent (29, ...) //nEventId=29
```

Description

This event gets fired whenever a view of a document becomes visible (i.e. becomes the active view) or invisible (i.e. another view becomes the active view or the document gets closed). However, the first view activation event after a document gets opened cannot be received, since there is no document object to get the event from. Use the [Application.OnDocumentOpened](#) event instead.

Application

See also

Property: [Application](#) as [Application](#) (read-only)

Description

Accesses the XMLSpy application object.

Errors

- 1400 The object is no longer valid.
- 1407 Invalid address for the return parameter was specified.

AssignDTD

See also

Method: [AssignDTD](#)(*strDTDFile* as String, *bDialog* as Boolean)

Description

The method places a reference to the DTD file "strDTDFile" into the document. Note that no error occurs if the file does not exist, or is not accessible. If *bDialog* is true XMLSpy presents a dialog to set the file.

See also Simple document access.

Errors

- 1400 The object is no longer valid.
- 1409 You are not allowed to assign a DTD to the document.

AssignSchema

See also

Method: [AssignSchema](#) (*strSchemaFile* as String, *bDialog* as Boolean)

Description

The method places a reference to the schema file "strSchemaFile" into the document. Note that

no error occurs if the file does not exist or is not accessible. If `bDialog` is true XMLSpy presents a dialog to set the file.

See also Simple document access.

Errors

- 1400 The object is no longer valid.
- 1409 You are not allowed to assign a schema file to the document.

AssignXSL**See also**

Method: `AssignXSL` (`strXSLFile` as String, `bDialog` as Boolean)

Description

The method places a reference to the XSL file "strXSLFile" into the document. Note that no error occurs if the file does not exist or is not accessible. If `bDialog` is true XMLSpy presents a dialog to set the file.

Errors

- 1400 The object is no longer valid.
- 1409 You are not allowed to assign an XSL file to the document.

AssignXSLFO**See also**

Method: `AssignXSLFO` (`strXSLFOFile` as String, `bDialog` as Boolean)

Description

The method places a reference to the XSLFO file "strXSLFile" into the document. Note that no error occurs if the file does not exist or is not accessible. If `bDialog` is true XMLSpy presents a dialog to set the file.

Errors

- 1400 The object is no longer valid.
- 1409 You are not allowed to assign an XSL file to the document.

AsXMLString**See also**

Property: `AsXMLString` as String

Description

This property can be used to get or set the document content.

Errors

- 1400 The document object is no longer valid.
- 1404 Cannot create XMLData object.
- 1407 View mode cannot be switched.

AuthenticView

See also

Method: [AuthenticView](#) as [AuthenticView](#) (read-only)

Description

Returns an object that gives access to properties and methods specific to Authentic view. The object returned is only valid if the current document is opened in Authentic view mode. The lifetime of an object ends with the next view switch. Any attempt to access objects or any of its children afterwards will result in an error indicating that the object is invalid.

[AuthenticView](#) and [DocEditView](#) both provide automation access to the Authentic view mode of XMLSpy. Functional overlap is intentional. A future version of Authentic View will include all functionality of [DocEditView](#) and its sub-objects, thereby making usage of [DocEditView](#) obsolete.

Errors

- 1400 The object is no longer valid.
- 1417 Document needs to be open in authentic view mode.

Examples

```
' -----
' XMLSpy scripting environment - VBScript
' secure access to authentic view object
' -----
Dim objDocument
Set objDocument = Application.ActiveDocument
If (Not objDocument Is Nothing) Then
    ' we have an active document, now check for view mode
    If (objDocument.CurrentViewMode <> spyViewAuthentic) Then
        If (Not objDocument.SwitchViewMode (spyViewAuthentic)) Then
            MsgBox "Active document does not support authentic view
mode"
        Else
            ' now it is safe to access the authentic view object
            Dim objAuthenticView
            Set objAuthenticView = objDocument.AuthenticView
            ' now use the authentic view object
        End If
    End If
Else
    MsgBox "No document is open"
End If
```

Close

See also

Method: [Close](#) ([bDiscardChanges](#) as Boolean)

Description

To close the document call this method. If [bDiscardChanges](#) is true and the document is modified, the document will be closed but not saved.

Errors

- 1400 The object is no longer valid.
- 1401 Document needs to be saved first.

ConvertDTDOrSchema

See also

Method: `ConvertDTDOrSchema` (*nFormat* as [SPYDTDSchemaFormat](#),
nFrequentElements as [SPYFrequentElements](#))

Parameters

nFormat

Sets the schema output format to DTD, or W3C.

nFrequentElements

Create complex elements as elements or complex types.

Description

`ConvertDTDOrSchema` takes an existing schema format and converts it into a different format. For a finer tuning of DTD / schema conversion, use [ConvertDTDOrSchemaEx](#).

Errors

- 1400 The object is no longer valid.
- 1412 Error during conversion.

ConvertDTDOrSchemaEx

See also

Method: `ConvertDTDOrSchemaEx` (*nFormat* as [SPYDTDSchemaFormat](#),
nFrequentElements as [SPYFrequentElements](#), *sOutputPath* as String,
nOutputPathDialogAction as [SPYDialogAction](#))

Parameters

nFormat

Sets the schema output format to DTD, or W3C.

nFrequentElements

Create complex elements as elements or complex types.

sOutputPath

The file path for the newly generated file.

nOutputPathDialogAction

Defines the dialog interaction for this call.

Description

`ConvertDTDOrSchemaEx` takes an existing schema format and converts it into a different format.

Errors

- 1400 The object is no longer valid.
- 1412 Error during conversion.

ConvertToWSDL20

Method: `ConvertToWSDL20` (*sFilePath* as String, *bShowDialogs* as Boolean)

Parameters

sFilePath

This specifies the file name of the converted WSDL. In case the source WSDL includes files which also must be converted, then only the directory part of the given path is used and the file names are generated automatically.

bShowDialogs

Defines whether file/folder selection dialogs are shown.

Description

Converts the WSDL 1.1 document to a WSDL 2.0 file. It will also convert any referenced WSDL files that are referenced from within this document. Note that this functionality is limited to WSDL View only. See [Document.CurrentViewMode](#). and [SPYViewModes](#).

Errors

- 1400 The document object is no longer valid.
- 1407 Invalid parameters have been passed or an empty file name has been specified as output target.
- 1417 The document is not opened in WSDL view, maybe it is not an '.wsdl' file.
- 1421 Feature is not available in this edition.
- 1433 WSDL 1.1 to WSDL 2.0 conversion failed.

CreateChild

See also

Method: [CreateChild](#) (*nKind* as [SPYXMLDataKind](#)) as [XMLData](#)

Return Value

The method returns the new [XMLData](#) object.

Description

To create a new [XMLData](#) object use the [CreateChild\(\)](#) method. See also [Using XMLData](#).

Errors

- 1400 The object is no longer valid.
- 1404 Cannot create [XMLData](#) object.
- 1407 Invalid address for the return parameter was specified.

CreateDBStructureFromXMLSchema

See also

Method: [CreateDBStructureFromXMLSchema](#) (*pDatabase* as [DatabaseConnection](#), *pTables* as [ElementList](#), *bDropTableWithExistingName* as Boolean) as [String](#)

Description

[CreateDBStructureFromXMLSchema](#) exports the given tables to the specified database. The function returns the SQL statements that were necessary to perform the changes.

See also [GetDBStructureList](#).

Errors

- 1429 Database selection missing.
- 1430 Document export failed.

CreateSchemaDiagram

See also

Method: `CreateSchemaDiagram` (*nKind* as [SPYSchemaDefKind](#), *strName* as String, *strFile* as String)

Return Value

None.

Description

The method creates a diagram of the schema type `strName` of kind `nKind` and saves the output file into `strFile`. Note that this functionality is limited to Schema View only. See [Document](#), [CurrentViewMode](#), and [SPYViewModes](#).

Errors

- 1400 The object is no longer valid.
- 1414 Failed to save diagram.
- 1415 Invalid schema definition type specified.

CurrentViewMode

See also

Method: `CurrentViewMode` as [SPYViewModes](#)

Description

The property holds the current view mode of the document. See also [Document](#), [SwitchViewMode](#).

Errors

- 1400 The object is no longer valid.
- 1407 Invalid address for the return parameter was specified.

DataRoot

See also

Property: `DataRoot` as [XMLData](#) (read-only)

Description

This property provides access to the document's first `XMLData` object of type `spyXMLDataElement`. This is typically the root element for all document content data. See [XMLSpyDocumentRootElement](#) to get the root element of the whole document including XML prolog data. If the [CurrentViewMode](#) is not `spyViewGrid` or `spyViewAuthentic` an [UpdateXMLData](#) may be necessary to get access to the latest [XMLData](#).

Errors

- 1400 The document object is no longer valid.
- 1407 Invalid address for the return parameter was specified.

DocEditView

See also

Method: [DocEditView](#) as [DocEditView](#)

Description

Holds a reference to the current Authentic View object.

Errors

- 1400 The object is no longer valid.
- 1407 Invalid address for the return parameter was specified.
- 1417 Document needs to be open in authentic view mode.

Encoding

See also

Property: [Encoding](#) as String

Description

This property provides access to the document's encoding value. However, this property can only be accessed when the document is opened in *spyViewGrid*, *spyViewText* or *spyViewAuthentic*. See [CurrentViewMode](#) on how to detect that a document's actual view mode.

This property makes the method [SetEncoding](#) obsolete.

Possible values are, for example:

8859-1,
8859-2,
ASCII, ISO-646,
850,
1252,
1255,
SHIFT-JIS, MS-KANJI,
BIG5, FIVE,
UTF-7,
UTF-8,
UTF-16

Errors

- 1400 The document object is no longer valid.
- 1407 Invalid address for the return parameter was specified.
- 1416 Operation not supported in current view mode.

EndChanges

See also

Method: [EndChanges](#)()

Description

Use the method `EndChanges` to display all changes since the call to [Document.StartChanges](#).

Errors

1400 The object is no longer valid.

ExecuteXQuery

See also

Method: `ExecuteXQuery` (*strXMLFileName* as String)

Description

Execute the XQuery statements contained in the document. Use the XML file specified as XML source for the transformation. If your XQuery script does not use an XML source set the parameter `strXMLFileName` to an empty string.

Errors

1400 The document object is no longer valid.

1423 XQuery transformation error.

1424 Not all files required for operation could be loaded. Most likely, the file specified in `strXMLFileName` does not exist or is not valid.

ExportToDatabase

See also

Method: `ExportToDatabase` (*pFromChild* as [XMLData](#), *pExportSettings* as [ExportSettings](#), *pDatabase* as [DatabaseConnection](#))

Description

`ExportToDatabase` exports the XML document starting with the element `pFromChild`. The parameter `pExportSettings` defines the behaviour of the export (see [Application.GetExportSettings](#)). The parameter `pDatabase` specifies the destination of the export (see [Application.GetDatabaseSettings](#)). [UpdateXMLData\(\)](#) might be indirectly needed as you have to pass the [XMLData](#) as parameter to this function.

Errors

1400 The object is no longer valid.

1407 Invalid parameter or invalid address for the return parameter was specified.

1416 Error during export.

1429 Database selection missing.

1430 Document export failed.

Example

```
Dim objDoc As Document
Set objDoc = objSpy.ActiveDocument

' set the behaviour of the export with ExportSettings
Dim objExpSettings As ExportSettings
Set objExpSettings = objSpy.GetExportSettings

' set the destination with DatabaseConnection
Dim objDB As DatabaseConnection
Set objDB = objSpy.GetDatabaseSettings
```

```

objDB.CreateMissingTables = True
objDB.CreateNew = True
objDB.File = "C:\Export.mdb"

objDoc.ExportToDatabase objDoc.RootElement, objExpSettings, objDB
If Err.Number <> 0 Then
    a = MsgBox("Error: " & (Err.Number - vbObjectError) & Chr(13) &
        "Description: " & Err.Description)
End If

```

ExportToText

See also

Method: `ExportToText` (*pFromChild* as [XMLData](#), *pExportSettings* as [ExportSettings](#), *pTextSettings* as [TextImportExportSettings](#))

Description

`ExportToText` exports tabular information from the document starting at `pFromChild` into one or many text files. Columns of the resulting tables are generated in alphabetical order of the column header names. Use [GetExportElementList](#) to learn about the data that will be exported. The parameter `pExportSettings` defines the specifics for the export. Set the property [ExportSettings.ElementList](#) to the - possibly modified - list returned by [GetExportElementList](#) to avoid exporting all contained tables. The parameter `pTextSettings` defines the options specific to text export and import. You need to set the property [TextImportExportSettings.DestinationFolder](#) before you call `ExportToText`. [UpdateXMLData\(\)](#) might be indirectly needed as you have to pass the [XMLData](#) as parameter to this function.

See also [Import and export of data](#).

Errors

- 1400 The object is no longer valid.
- 1407 Invalid parameter or invalid address for the return parameter was specified.
- 1416 Error during export.
- 1430 Document export failed.

Example

```

' -----
' VBA client code fragment - export document to text files
' -----

Dim objDoc As Document
Set objDoc = objSpy.ActiveDocument

Dim objExpSettings As ExportSettings
Set objExpSettings = objSpy.GetExportSettings
objExpSettings.ElementList = objDoc.GetExportElementList(
    objDoc.RootElement,
    objExpSettings)

Dim objTextExp As TextImportExportSettings
Set objTextExp = objSpy.GetTextExportSettings
objTextExp.HeaderRow = True
objTextExp.DestinationFolder = "C:\Exports"

On Error Resume Next
objDoc.ExportToText objDoc.RootElement, objExpSettings, objTextExp

If Err.Number <> 0 Then

```

```
    a = MsgBox("Error: " & (Err.Number - vbObjectError) & Chr(13) &  
    "Description: " & Err.Description)  
    End If
```

FullName

See also

Property: [FullName](#) as String

Description

This property can be used to get or set the full file name - including the path - to where the document gets saved. The validity of the name is not verified before the next save operation.

This property makes the methods [GetPathName](#) and [SetPathName](#) obsolete.

Errors

- 1400 The document object is no longer valid.
- 1402 Empty string has been specified as full file name.

GenerateDTDOrSchema

See also

Method: [GenerateDTDOrSchema](#) (*nFormat* as [SPYDTDSchemaFormat](#), *nValuesList* as integer, *nDetection* as [SPYTypeDetection](#), *nFrequentElements* as [SPYFrequentElements](#))

Parameters

nFormat

Sets the schema output format to DTD, or W3C.

nValuesList

Generate not more than this amount of enumeration-facets per type. Set to -1 for unlimited.

nDetection

Specifies granularity of simple type detection.

nFrequentElements

Shall the types for all elements be defined as global? Use that value *spyGlobalComplexType* to define them on global scope. Otherwise, use the value *spyGlobalElements*.

Description

Use this method to automatically generate a DTD or schema for the current XML document.

For a finer tuning of DTD / schema generation, use [GenerateDTDOrSchemaEx](#).

Note that this functionality is not available in ZIP View only. See

[Document.CurrentViewMode](#) and [SPYViewModes](#).

Errors

- 1400 The object is no longer valid.
- 1407 Invalid parameter or invalid address for the return parameter was specified.

GenerateDTDOrSchemaEx

See also

Method: [GenerateDTDOrSchemaEx](#) (*objDlg* as [DTDSchemaGeneratorDlg](#)) as [Document](#)

Description

Use this method to automatically generate a DTD or schema for the current XML document. A [DTDSchemaGeneratorDlg](#) object is used to pass information to the schema/DTD generator. The generation process can be configured to allow user interaction or run without further user input.

Note that this functionality is not available in ZIP View only. See

[Document.CurrentViewMode](#). and [SPYViewModes](#).

Errors

- 1400 The object is no longer valid.
- 1407 Invalid parameter or invalid address for the return parameter was specified.

GenerateProgramCode

Method: [GenerateProgramCode](#) (*objDlg* as [CodeGeneratorDlg](#))

Only available/enabled in the Enterprise edition. An error is returned, if accessed by any other version.

Description

Generate Java, C++ or C# class files from the XML Schema definitions in your document. A [CodeGeneratorDlg](#) object is used to pass information to the code generator. The generation process can be configured to allow user interaction or run without further user input.

Errors

- 1400 The document object is no longer valid.
- 1407 An empty file name has been specified.
- 1421 Feature not available in this edition

GenerateSampleXML

Method: [GenerateSampleXML](#) (*objDlg* as [GenerateSampleXMLDlg](#)) as [Document](#)

Description

Generates a sample XML if the document is a schema or DTD. Use [Dialogs.GenerateSampleXMLDlg](#) to get an initialized set of options.

Available with TypeLibrary version 1.5

Errors

- 1400 The document object is no longer valid.

GenerateSchemaDocumentation

Method: [GenerateSchemaDocumentation](#) (*objDlg* as [SchemaDocumentationDlg](#))

Description

Generate documentation for a schema definition file in HTML, MS-Word, or RTF format. The

parameter `objDlg` is used to parameterize the generation process. Use [Dabgs.SchemaDocumentationDlg](#) to get an initialized set of options. As a minimum, you will need to set the property [SchemaDocumentationDlg.OutputFile](#) before starting the generation process. Note that this functionality is limited to Schema View only. See [Document.CurrentViewMode](#) and [SPYViewModes](#).

Errors

- 1400 The document object is no longer valid.
- 1407 Invalid parameters have been passed or an empty file name has been specified as output target.
- 1417 The document is not opened in schema view, maybe it is not an '.xsd' file.
- 1421 Feature is not available in this edition.
- 1422 Error during generation

GenerateWSDL20Documentation

Method: [GenerateWSDL20Documentation](#) (`objDlg` as [WSDL20DocumentationDlg](#))

Description

Generate documentation for a WSDL definition file in HTML, MS-Word, or RTF format. The parameter `objDlg` is used to parameterize the generation process. Use [Dabgs.WSDL20DocumentationDlg](#) to get an initialized set of options. As a minimum, you will need to set the property [WSDL20DocumentationDlg.OutputFile](#) before starting the generation process. Note that this functionality is limited to WSDL View only. See [Document.CurrentViewMode](#) and [SPYViewModes](#).

Errors

- 1400 The document object is no longer valid.
- 1407 Invalid parameters have been passed or an empty file name has been specified as output target.
- 1417 The document is not opened in schema view, maybe it is not an '.xsd' file.
- 1421 Feature is not available in this edition.
- 1422 Error during generation

GenerateWSDLDocumentation

Method: [GenerateWSDLDocumentation](#) (`objDlg` as [WSDLDocumentationDlg](#))

Description

Generate documentation for a WSDL definition file in HTML, MS-Word, or RTF format. The parameter `objDlg` is used to parameterize the generation process. Use [Dabgs.WSDLDocumentationDlg](#) to get an initialized set of options. As a minimum, you will need to set the property [WSDLDocumentationDlg.OutputFile](#) before starting the generation process. Note that this functionality is limited to WSDL View only. See [Document.CurrentViewMode](#) and [SPYViewModes](#).

Errors

- 1400 The document object is no longer valid.
- 1407 Invalid parameters have been passed or an empty file name has been specified as output target.
- 1417 The document is not opened in schema view, maybe it is not an '.xsd' file.
- 1421 Feature is not available in this edition.

1422 Error during generation

GenerateXBRLDocumentation

Method: `GenerateXBRLDocumentation` (*objDlg* as [XBRLDocumentationDlg](#))

Description

Generate documentation for a WSDL definition file in HTML, MS-Word, or RTF format. The parameter *objDlg* is used to parameterize the generation process. Use [Dabcs.XBRLDocumentationDlg](#) to get an initialized set of options. As a minimum, you will need to set the property [XBRLDocumentationDlg.OutputFile](#) before starting the generation process. Note that this functionality is limited to XBRL View only. See [Document.CurrentViewMode](#) and [SPYViewModes](#).

Errors

- 1400 The document object is no longer valid.
- 1407 Invalid parameters have been passed or an empty file name has been specified as output target.
- 1417 The document is not opened in schema view, maybe it is not an '.xsd' file.
- 1421 Feature is not available in this edition.
- 1422 Error during generation

GetDBStructureList

See also

Method: `GetDBStructureList` (*pDatabase* as [DatabaseConnection](#)) as [ElementList](#)

Description

`GetDBStructureList` creates a collection of elements from the Schema document for which tables in the specified database are created. The function returns a collection of [ElementListItems](#) where the properties [ElementListItem.Name](#) contain the names of the tables.

See also [CreateDBStructureFromXMLSchema](#).

Errors

- 1400 The object is no longer valid.
- 1427 Failed creating parser for the specified XML.
- 1428 Export of element list failed.
- 1429 Database selection missing.

GetExportElementList

See also

Method: `GetExportElementList` (*pFromChild* as [XMLData](#), *pExportSettings* as [ExportSettings](#)) as [ElementList](#)

Description

`GetExportElementList` creates a collection of elements to export from the document,

depending on the settings in `pExportSettings` and starting from the element `pFromChild`. The function returns a collection of `ElementListItems` where the properties `ElementListItem.Name` contain the names of the tables that can be exported from the document. The property `ElementListItem.FieldCount` contains the number of columns in the table. The property `ElementListItem.RecordCount` contains the number of records in the table. The property `ElementListItem.ElementKind` is unused. `UpdateXMLData()` might be indirectly needed as you have to pass the `XMLData` as parameter to this function.

See also [Import and export of data](#).

Errors

- 1400 The object is no longer valid.
- 1407 Invalid parameter or invalid address for the return parameter was specified.
- 1427 Failed creating parser for the specified XML.
- 1428 Export of element list failed.

GetPathName (obsolete)

Superseded by [Document.FullName](#)

```
// ----- javascript sample -----  
// instead of:  
// strPathName = Application.ActiveDocument.GetPathName();  
// use now:  
strPathName = Application.ActiveDocument.FullName;
```

See also

Method: `GetPathName()` as String

Description

The method `GetPathName` gets the path of the active document.

See also [Document.SetPathName](#) (obsolete).

GridView

See also

Property: `GridView` as [GridView](#)

Description

This property provides access to the grid view functionality of the document.

Errors

- 1400 The object is no longer valid.
- 1407 Invalid address for the return parameter was specified.
- 1417 Document needs to be open in enhanced grid view mode.

IsModified**See also**

Property: [IsModified](#) as Boolean

Description

True if the document is modified.

Errors

- 1400 The object is no longer valid.
- 1407 Invalid address for the return parameter was specified.

IsValid**See also**

Method: [IsValid](#) (*strError* as Variant, *nErrorPos* as Variant, *pBadData* as Variant) as Boolean

Return Value

True if the document is valid, false if not.

Description

IsValid() validates the document against its associated schema or DTD. strError gives you the same error message as XMLSpy, if you validate the file within the editor.

nErrorPos is obsolete and only present for compatibility reasons. pBadData is a reference to the [XMLData](#) object raising the error. It is only available if the document is currently displayed in TextView, GridView or AuthenticView.

Errors

- 1400 The object is no longer valid.
- 1407 Invalid parameter or invalid address for the return parameter was specified.
- 1408 Unable to validate file.

Examples

For an example written in VBA see Overview - Simple document access

```
' -----
' XMLSpy scripting environment - VBScript
' -----
Dim errorText
Dim errorPos
Dim badData

' validate the active document
Set doc = Application.ActiveDocument
valid = doc.IsValid(errorText, errorPos, badData)

If (Not valid) Then
    ' create the validation error message text
    Text = errorText

    If ( Not IsEmpty( badData) ) Then
```



```

        Text = Text & "(" & badData.Name & "/" & badData.TextValue & ")"
    End If

    Call MsgBox("validation error[" & errorPos & "]: " & Text)
End If

// -----
// XMLSpy scripting environment - JScript
// -----
// define as arrays to support their usage as return parameters
var errorText = new Array(1);
var errorPos = new Array(1);
var badData = new Array(1);

// validate the active document
var doc = Application.ActiveDocument;
var valid = doc.IsValid(errorText, errorPos, badData);

if (! valid)
{
    // compose the error description
    var text = errorText;

    // access that XMLData object only if filled in
    if (badData[0] != null)
        text += "(" & badData[0].Name & "/" & badData[0].TextValue & ")"
;

    MsgBox("validation error[" & errorPos & "]: " & text);
}

```

IsWellFormed

See also

Method: `IsWellFormed` (*pData* as `XMLData`, *bWithChildren* as `Boolean`, *strError* as `Variant`, *nErrorPos* as `Variant`, *pBadXMLData* as `Variant`) as `Boolean`

Return Value

True if the document is well formed.

Description

`IsWellFormed` checks the document for well-formedness starting at the element `pData`.

If the document is not well formed, `strError` contains an error message, `nErrorPos` the position in the file and `pBadXMLData` holds a reference to the element which breaks the well-formedness. These out-parameters are defined as `VARIANTs` to support scripting languages like `VBScript`.

Errors

- 1400 The object is no longer valid.
- 1407 Invalid parameter or invalid address for the return parameter was specified.

Example

See [IsValid](#).

Name**See also**

Property: [Name](#) as String (read-only)

Description

Use this property to retrieve the name - not including the path - of the document file. To change the file name for a document use the property [FullName](#).

Errors

- 1400 The document object is no longer valid.
- 1407 Invalid address for the return parameter was specified.

Parent**See also**

Property: [Parent](#) as [Documents](#) (read-only)

Description

Access the parent of the document object.

Errors

- 1400 The document object is no longer valid.
- 1407 Invalid address for the return parameter was specified.

Property: [Parent](#) as [Application](#) (read-only)

Path**See also**

Property: [Path](#) as String (read-only)

Description

Use this property to retrieve the path - not including the file name - of the document file. To change the file name and path for a document use the property [FullName](#).

Errors

- 1400 The document object is no longer valid.
- 1407 Invalid address for the return parameter was specified.

RootElement**See also**

Property: [RootElement](#) as [XMLData](#) (read-only)

Description

The property `RootElement` provides access to the root element of the XML structure of the document including the XML prolog data. To access the first element of a document's content navigate to the first child of kind `spyXMLDataElement` or use the [Document.DataRoot](#) property. If the [CurrentViewMode](#) is not `spyViewGrid` or `spyViewAuthentic` an [UpdateXMLData](#) may be

necessary to get access to the latest [XMLData](#).

Errors

- 1400 The document object is no longer valid.
- 1407 Invalid address for the return parameter was specified.

Save**See also**

Method: [Save](#)()

Description

The method writes any modifications of the document to the associated file. See also [Document.FullName](#).

Errors

- 1400 The document object is no longer valid.
- 1407 An empty file name has been specified.
- 1403 Error when saving file, probably the file name is invalid.

SaveAs**See also**

Method: [SaveAs](#) (*strFileName* as String)

Description

Save the document to the file specified. If saving was successful, the [FullName](#) property gets set to the specified file name.

Errors

- 1400 The document object is no longer valid.
- 1407 An empty file name has been specified.
- 1403 Error when saving file, probably the file name is invalid.

Saved**See also**

Property: [Saved](#) as Boolean (read-only)

Description

This property can be used to check if the document has been saved after the last modifications. It returns the negation of [IsModified](#).

Errors

- 1400 The document object is no longer valid.
- 1407 Invalid address for the return parameter was specified.

SaveInString**See also**

Method: `SaveInString` (*pData* as [XMLData](#), *bMarked* as Boolean) as String

Parameters

pData

[XMLData](#) element to start. Set *pData* to [Document.RootElement](#) if you want to copy the complete file.

bMarked

If *bMarked* is true, only the elements selected in the grid view are copied.

Return Value

Returns a string with the XML data.

Description

`SaveInString` starts at the element *pData* and converts the [XMLData](#) objects to a string representation. [UpdateXMLData\(\)](#) might be indirectly needed as you have to pass the [XMLData](#) as parameter to this function.

Errors

- 1400 The object is no longer valid.
- 1407 Invalid parameter or invalid address for the return parameter was specified.

SaveToURL

See also

Method: `SaveToURL` (*strURL* as String, *strUser* as String, *strPassword* as String)

Return Value

Description

`SaveToURL()` writes the document to the URL *strURL*. This method does not set the permanent file path of the document.

Errors

- 1400 The object is no longer valid.
- 1402 Invalid URL specified.
- 1403 Error while saving to URL.

SetActiveDocument

See also

Method: `SetActiveDocument()`

Description

The method sets the document as the active and brings it to the front.

Errors

- 1400 The object is no longer valid.

SetEncoding (obsolete)

Superseded by [Document.Encoding](#)

```
// ----- javascript sample -----  
// instead of:  
// Application.ActiveDocument.SetEncoding("UTF-16");  
// use now:  
Application.ActiveDocument.Encoding = "UTF-16";
```

See also

Method: [SetEncoding](#) (*strEncoding* as String)

Description

`SetEncoding` sets the encoding of the document like the menu item "File/Encoding..." in XMLSpy. Possible values for `strEncoding` are, for example:

- 8859-1,
- 8859-2,
- ASCII, ISO-646,
- 850,
- 1252,
- 1255,
- SHIFT-JIS, MS-KANJI,
- BIG5, FIVE,
- UTF-7,
- UTF-8,
- UTF-16

SetExternallsValid

See also

Method: `SetExternallsValid` (*bValid* as Boolean)

Parameters

bValid

Sets the result of an external validation process.

Description

The internal information set by this method is only queried on cancelling the default validation in any [OnBeforeValidate](#) handler.

Available with TypeLibrary version 1.5

Errors

1400 The object is no longer valid.

SetPathName (obsolete)

Superseded by [Document.FullName](#)

```
// ----- javascript sample -----  
// instead of:  
// Application.ActiveDocument.SetPathName("C:\\myXMLFiles\\test.xml");  
// use now:  
Application.ActiveDocument.FullName = "C:\\myXMLFiles\\test.xml";
```

See also

Method: `SetPathName` (*strPath* as String)

Description

The method `SetPathName` sets the path of the active document. `SetPathName` only copies the string and does not check if the path is valid. All succeeding save operations are done into this file.

StartChanges

See also

Method: `StartChanges`()

Description

After `StartChanges` is executed XMLSpy will not update its editor windows until [Document.EndChanges](#) is called. This increases performance of complex tasks to the XML structure.

Errors

1400 The object is no longer valid.

Suggestions

Property: [Suggestions](#) as Array

Description

This property contains the last valid user suggestions for this document. The XMLSpy generated suggestions can be modified before they are shown to the user in the [OnBeforeShowSuggestions](#) event.

Errors

- 1400 The object is no longer valid.
- 1407 Invalid parameter or invalid address for the return parameter was specified.

SwitchViewMode

See also

Method: [SwitchViewMode](#) (*nMode* as [SPYViewModes](#)) as Boolean

Return value

Returns true if view mode is switched.

Description

The method sets the current view mode of the document in XMLSpy. See also [Document.CurrentViewMode](#).

Errors

- 1400 The object is no longer valid.
- 1407 Invalid address for the return parameter was specified.
- 1417 Invalid view mode specified.

TextView

See also

Property: [TextView](#) as [TextView](#)

Description

This property provides access to the text view functionality of the document.

Errors

- 1400 The object is no longer valid.
- 1407 Invalid address for the return parameter was specified.

Title

See also

Property: [Title](#) as String (read-only)

Description

[Title](#) contains the file name of the document. To get the path and filename of the file use [FullName](#).

Errors

- 1400 The document object is no longer valid.
- 1407 Invalid address for the return parameter was specified.

TransformXSL**See also**

Method: [TransformXSL\(\)](#)

Description

`TransformXSL` processes the XML document via the associated XSL file. See [Document.AssignXSL](#) on how to place a reference to a XSL file into the document.

Errors

- 1400 The document object is no longer valid.
- 1411 Error during transformation process.

TransformXSLEx**See also**

Method: `TransformXSLEx(nAction as SPYDialogAction)`

Description

`TransformXSLEx` processes the XML document via the associated XSL file. The parameter specifies whether a dialog asking for the result document name should pop up or not. See [Document.AssignXSL](#) on how to place a reference to a XSL file into the document.

Errors

- 1400 The document object is no longer valid.
- 1411 Error during transformation process.

TransformXSLFO**See also**

Method: [TransformXSLFO\(\)](#)

Description

`TransformXSLFO` processes the XML document via the associated XSLFO file. See [AssignXSLFO](#) on how to place a reference to a XSLFO file into the document. You need to assign a FOP processor to XMLSpy before you can use this method.

Errors

- 1400 The document object is no longer valid.
- 1411 Error during transformation process.

TreatXBRLInconsistencies AsErrors

Property: [TreatXBRLInconsistenciesAsErrors](#) as Boolean

Description

If this is set to `true` the `Document.IsValid()` method will return `false` for XBRL instances containing inconsistencies as defined by the XBRL Specification. The default value of this

property is `false`.

Errors

- 1400 The document object is no longer valid.
- 1407 Invalid address for the return parameter was specified.

UpdateViews**See also**

Method: [UpdateViews\(\)](#)

Description

To redraw the Enhanced Grid View and the Tree View call `UpdateViews`. This can be important after you changed the `XMLData` structure of a document. This method does not redraw the text view of XMLSpy.

Errors

- 1400 The document object is no longer valid.

UpdateXMLData**See also**

Method: [UpdateXMLData\(\)](#) as Boolean

Description

The [XMLData](#) tree is updated from the current view. Please note that this can fail in case of the Text View if the current XML text is not well-formed. This is not necessary if [CurrentViewMode](#) is `spyViewGrid` or `spyViewAuthentic` because these views keep the [XMLData](#) updated.

Available with TypeLibrary version 1.5

Errors

- 1400 The document object is no longer valid.

3.2.11 Documents

See also**Properties**

[Count](#)
[Item](#)

Methods

[NewAuthenticFile](#)
[NewFile](#)
[NewFileFromText](#)
[OpenAuthenticFile](#)
[OpenFile](#)
[OpenURL](#)
[OpenURLDialog](#)

Description

This object represents the set of documents currently open in XMLSpy. Use this object to open further documents or iterate through already opened documents.

Examples

```
' -----
' XMLSpy scripting environment - VBScript
' iterate through open documents
' -----

Dim objDocuments
Set objDocuments = Application.Documents

For Each objDoc In objDocuments
    'do something useful with your document
    objDoc.SetActiveDocument()
Next

// -----
// XMLSpy scripting environment - JScript
// close all open documents
// -----
for ( var iter = new Enumerator ( Application.Documents );
      ! iter.atEnd();
      iter.moveNext() )
{
    // MsgBox ("Closing file " + iter.item().Name);
    iter.item().Close ( true );
}
```

Count**See also**

Property: [Count](#) as long

Description

Count of open documents.

Errors

- 1600 Invalid Documents object
- 1601 Invalid input parameter

Item**See also**

Method: [Item](#) (*n* as long) as [Document](#)

Description

Gets the document with the index *n* in this collection. Index is 1-based.

Errors

- 1600 Invalid Documents object
- 1601 Invalid input parameter

NewAuthenticFile

See also

Method: `NewAuthenticFile` (*strSPSPath* as String, *strXMLPath* as String) as [Document](#)

Parameters

strSPSPath

The path to the SPS document.

strXMLPath

The new XML document name.

Return Value

The method returns the new document.

Description

`NewAuthenticFile` creates a new XML file and opens it in Authentic View using SPS design *strSPSPath*.

NewFile

See also

Method: `NewFile` (*strFile* as String, *strType* as String) as [Document](#)

Parameters

strFile

Full path of new file.

strType

Type of new file as string (i.e. "xml", "xsd", ...)

Return Value

Returns the new file.

Description

`NewFile` creates a new file of type *strType* (i.e. "xml"). The newly created file is also the `ActiveDocument`.

NewFileFromText

See also

Method: `NewFileFromText` (*strText* as String, *strType* as String) as [Document](#)

Parameters

strText

The content of the new document in plain text.

strType

Type of the document to create (i.e. "xml").

Return Value

The method returns the new document.

Description

NewFileFromText creates a new document with strText as its content.

OpenAuthenticFile**See also**

Method: `OpenAuthenticFile` (*strSPSPath* as String, *strXMLPath* as String) as [Document](#)

Parameters

strSPSPath

The path to the SPS document.

strXMLPath

The path to the XML document (can be empty).

Return Value

The method returns the new document.

Description

OpenAuthenticFile opens an XML file or database in Authentic View using SPS design strSPSPath.

OpenFile**See also**

Method: `OpenFile` (*strPath* as String, *bDialog* as Boolean) as [Document](#)

Parameters

strPath

Path and file name of file to open.

bDialog

Show dialogs for user input.

Return Value

Returns the opened file on success.

Description

`OpenFile` opens the file *strPath* . If *bDialog* is TRUE, a file-dialog will be displayed.

Example

```
Dim objDoc As Document
Set objDoc = objSpy.Documents.OpenFile(strFile, False)
```

OpenURL**See also**

Method: `OpenURL` (*strURL* as String, *nURLType* as [SPYURLTypes](#), *nLoading* as [SPYLoading](#), *strUser* as String, *strPassword* as String) as [Document](#)

Parameters

`strURL`

URL to open as document.

`nURLType`

Type of document to open. Set to -1 for auto detection.

`nLoading`

Set `nLoading` to 0 (zero) if you want to load it from cache or proxy. Otherwise set `nLoading` to 1.

`strUser`

Name of the user if required. Can be empty.

`strPassword`

Password for authentication. Can be empty.

Return Value

The method returns the opened document.

Description

`OpenURL` opens the URL `strURL`.

OpenURLDialog**See also**

Method: `OpenURLDialog` (`strURL` as String, `nURLType` as [SPYURLTypes](#), `nLoading` as [SPYLoading](#), `strUser` as String, `strPassword` as String) as [Document](#)

Parameters

`strURL`

URL to open as document.

`nURLType`

Type of document to open. Set to -1 for auto detection.

`nLoading`

Set `nLoading` to 0 (zero) if you want to load it from cache or proxy. Otherwise set `nLoading` to 1.

`strUser`

Name of the user if required. Can be empty.

`strPassword`

Password for authentication. Can be empty.

Return Value

The method returns the opened document.

Description

`OpenURLDialog` displays the "open URL" dialog to the user and presets the input fields with the given parameters.

3.2.12 DTDSchemaGeneratorDlg

See also

Properties and Methods

Standard automation properties

[Application](#)

[Parent](#)

[DTDSchemaFormat](#)

[ValueList](#)

[TypeDetection](#)

[FrequentElements](#)

[MergeAllEquaNamed](#)

[ResolveEntities](#)

[AttributeTypeDefinition](#)

[GlobalAttributes](#)

[OnlyStringEnums](#)

[MaxEnumLength](#)

[OutputPath](#)

[OutputPathDialogAction](#)

Description

Use this object to configure the generation of a schema or DTD. The method [GenerateDTDOrSchemaEx](#) expects a [DTDSchemaGeneratorDlg](#) as parameter to configure the generation as well as the associated user interactions.

Application

Property: [Application](#) as [Application](#) (read-only)

Description

Access the XMLSpy application object.

Errors

- 3000 The object is no longer valid.
- 3001 Invalid address for the return parameter was specified.

AttributeTypeDefinition

Property: [AttributeTypeDefinition](#) as [SPYAttributeTypeDefinition](#)

Description

Specifies how attribute definitions get merged.

Errors

- 3000 The object is no longer valid.
- 3001 Invalid address for the return parameter was specified.

DTDSchemaFormat

Property: [DTDSchemaFormat](#) as [SPYDTDSchemaFormat](#)

Description

Sets the schema output format to DTD, or W3C.

Errors

- 3000 The object is no longer valid.
- 3001 Invalid address for the return parameter was specified.

FrequentElements

Property: *FrequentElements* as [SPYFrequentElements](#)

Description

Shall the types for all elements be defined as global? Use that value *spyGlobalComplexType* to define them on global scope. Otherwise, use the value *spyGlobalElements*.

Errors

- 3000 The object is no longer valid.
- 3001 Invalid address for the return parameter was specified.

GlobalAttributes

Property: *GlobalAttributes* as Boolean

Description

Shall attributes with same name and type be resolved globally?

Errors

- 3000 The object is no longer valid.
- 3001 Invalid address for the return parameter was specified.

MaxEnumLength

Property: *MaxEnumLength* as Integer

Description

Specifies the maximum number of characters allowed for enumeration names. If one value is longer than this, no enumeration will be generated.

Errors

- 3000 The object is no longer valid.
- 3001 Invalid address for the return parameter was specified.

MergeAllEqualNamed

Property: *MergeAllEqualNamed* as Boolean

Description

Shall types of all elements with the same name be merged into one type?

Errors

- 3000 The object is no longer valid.
- 3001 Invalid address for the return parameter was specified.

OnlyStringEnums

Property: [OnlyStringEnums](#) as Boolean

Description

Specifies if enumerations will be created only for plain strings or all types of values.

Errors

- 3000 The object is no longer valid.
- 3001 Invalid address for the return parameter was specified.

OutputPath

Property: [OutputPath](#) as String

Description

Selects the file name for the generated schema/DTD.

Errors

- 3000 The object is no longer valid.
- 3001 Invalid address for the return parameter was specified.

OutputPathDialogAction

Property: [OutputPathDialogAction](#) as [SPYDialogAction](#)

Description

Defines how the sub-dialog for selecting the schema/DTD output path gets handled. Set this value to *spyDialogUserInput(2)* to show the dialog with the current value of the [OutputPath](#) property as default. Use *spyDialogOK(0)* to hide the dialog from the user.

Errors

- 3000 The object is no longer valid.
- 3001 Invalid address for the return parameter was specified.

Parent

Property: [Parent](#) as [Dialogs](#) (read-only)

Description

Access the parent of the object.

Errors

- 3000 The object is no longer valid.
- 3001 Invalid address for the return parameter was specified.

ResolveEntities

Property: [ResolveEntities](#) as Boolean

Description

Shall all entities be resolved before generation starts? If yes, an info-set will be built.

Errors

- 3000 The object is no longer valid.
- 3001 Invalid address for the return parameter was specified.

TypeDetection

Property: [TypeDetection](#) as [SPYTypeDetection](#)

Description

Specifies granularity of simple type detection.

Errors

- 3000 The object is no longer valid.
- 3001 Invalid address for the return parameter was specified.

ValueList

Property: [ValueList](#) as Integer

Description

Generate not more than this amount of enumeration-facets per type. Set to -1 for unlimited.

Errors

- 3000 The object is no longer valid.
- 3001 Invalid address for the return parameter was specified.

3.2.13 ElementList

See also

Properties

[Count](#)
[Item](#)

Methods

[RemoveElement](#)

Description

Element lists are used for different purposes during export and import of data. Depending on this purpose, different properties of [ElementListItem](#) are used.

It can hold

- a list of table names returned by a call to [Application.GetDatabaseTables](#),
- a list of field names returned by a call to [Application.GetDatabaseImportElementList](#) or [Application.GetTextImportElementList](#),
- a field name filter list used in [Application.ImportFromDatabase](#) and [Application.ImportFromText](#),
- a list of table names and counts for their rows and columns as returned by calls to [GetExportElementList](#) or
- a field name filter list used in [Document.ExportToDatabase](#) and [Document.ExportToText](#).

Count**See also**

Property: [Count](#) as long (read-only)

Description

Count of elements in this collection.

Item**See also**

Method: [Item](#)([n](#) as long) as [ElementListItem](#)

Description

Gets the element with the index [n](#) from this collection. The first item has index 1.

RemoveElement**See also**

Method: [RemoveElement](#)([Index](#) as long)

Description

[RemoveElement](#) removes the element [Index](#) from the collection. The first [Item](#) has index 1.

3.2.14 ElementListItem**See also****Properties**

[Name](#)

[ElementKind](#)

[FieldCount](#)

[RecordCount](#)

Description

An element in an [ElementList](#). Usage of its properties depends on the purpose of the element list. For details see [ElementList](#).

ElementKind**See also**

Property: [ElementKind](#) as [SPYXMLDataKind](#)

Description

Specifies if a field should be imported as XML element (data value of `spyXMLDataElement`) or attribute (data value of `spyXMLDataAttr`).

FieldCount**See also**

Property: [FieldCount](#) as long (read-only)

Description

Count of fields (i.e. columns) in the table described by this element. This property is only valid after a call to [DocumentGetExportElementList](#).

Name**See also**

Property: [Name](#) as String (read-only)

Description

Name of the element. This is either the name of a table or a field, depending on the purpose of the element list.

RecordCount**See also**

Property: [RecordCount](#) as long (read-only)

Description

Count of records (i.e. rows) in the table described by this element. This property is only valid after a call to [DocumentGetExportElementList](#).

3.2.15 ExportSettings

See also**Properties**

[ElementList](#)

[EntitiesToText](#)

[ExportAllElements](#)

[SubLevelLimit](#)

[FromAttributes](#)

[FromSingleSubElements](#)

[FromTextValues](#)

[CreateKeys](#)

[IndependentPrimaryKey](#)

[Namespace](#)

[ExportCompleteXML](#)

[StartFromElement](#)

Description

[ExportSettings](#) contains options used during export of XML data to a database or text file. See [Import and export of data](#) for a general overview.

CreateKeys**See also**

Property: [CreateKeys](#) as Boolean

Description

This property turns creation of keys (i.e. primary key and foreign key) on or off. Default is True.

ElementList**See also**

Property: [ElementList](#) as [ElementList](#)

Description

Default is empty list. This list of elements defines which fields will be exported. To get the list of available fields use [Document.GetExportElementList](#). It is possible to prevent exporting columns by removing elements from this list with [ElementList.RemoveElement](#) before passing it to [Document.ExportToDatabase](#) or [Document.ExportToText](#).

EntitiesToText**See also**

Property: [EntitiesToText](#) as Boolean

Description

Defines if XML entities should be converted to text or left as they are during export. Default is True.

ExportAllElements**See also**

Property: [ExportAllElements](#) as Boolean

Description

If set to `true`, all elements in the document will be exported. If set to `false`, then [ExportSettings.SubLevelLimit](#) is used to restrict the number of sub levels to export. Default is `true`.

ExportCompleteXML**See also**

Property: [ExportCompleteXML](#) as Boolean

Description

Defines whether the complete XML is exported or only the element specified by

[StartFromElement](#) and its children. Default is True.

FromAttributes

See also

Property: [FromAttributes](#) as Boolean

Description

[SetFromAttributes](#) to false if no export data should be created from attributes. Default is True.

FromSingleSubElements

See also

Property: [FromSingleSubElements](#) as Boolean

Description

[SetFromSingleSubElements](#) to false if no export data should be created from elements. Default is True.

FromTextValues

See also

Property: [FromTextValues](#) as Boolean

Description

[SetFromTextValues](#) to false if no export data should be created from text values. Default is True.

IndependentPrimaryKey

See also

Property: [IndependentPrimaryKey](#) as Boolean

Description

Turns creation of independent primary key counter for every element on or off. If [ExportSettings.CreateKeys](#) is False, this property will be ignored. Default is True.

Namespace

See also

Property: [Namespace](#) as [SPYExportNamespace](#)

Description

The default setting removes all namespace prefixes from the element names. In some database formats the colon is not a legal character. Default is `spyNoNamespace`.

StartFromElement

See also

Property: [StartFromElement](#) as String

Description

Specifies the start element for the export. This property is only considered when [ExportCompleteXML](#) is false.

SubLevelLimit

See also

Property: [SubLevelLimit](#) as Integer

Description

Defines the number of sub levels to include for the export. Default is 0. This property is ignored if [ExportSettings.ExportAllElements](#) is true.

3.2.16 FileSelectionDlg

See also

Properties and Methods

Standard automation properties

[Application](#)

[Parent](#)

Dialog properties

[FullName](#)

Acceptance or cancellation of action that caused event

[DialogAction](#)

Description

The dialog object allows you to receive information about an event and pass back information to the event handler in the same way as with a user dialog. Use the [FileSelectionDlg.FullName](#) to select or modify the file path and set the [FileSelectionDlg.DialogAction](#) property to cancel or agree with the action that caused the event.

Application

See also

Property: [Application](#) as [Application](#) (read-only)

Description

Access the XMLSpy application object.

Errors

- 2400 The object is no longer valid.
- 2401 Invalid address for the return parameter was specified.

DialogAction

Property: [DialogAction](#) as [SPYDialogAction](#)

Description

If you want your script to perform the file selection operation without any user interaction necessary, simulate user interaction by either setting the property to *spyDialogOK(0)* or *spyDialogCancel(1)*.

To allow your script to fill in the default values but let the user see and react on the dialog, use the value *spyDialogUserInput(2)*. If you receive a `FileSelectionDlg` object in an event handler, *spyDialogUserInput(2)* is not supported and will be interpreted as *spyDialogOK(0)*.

Errors

- 2400 The object is no longer valid.
- 2401 Invalid value for dialog action or invalid address for the return parameter was specified.

FullName

Property: [FullName](#) as String

Description

Access the full path of the file the gets selected by the dialog. Most events that pass a `FileSelectionDlg` object to you allow you modify this value and thus influence the action that caused the event (e.g. load or save to a different location).

Errors

- 2400 The object is no longer valid.
- 2401 Invalid address for the return parameter was specified.

Parent

See also

Property: [Parent](#) as [Dialogs](#) (read-only)

Description

Access the parent of the object.

Errors

- 2400 The object is no longer valid.
- 2401 Invalid address for the return parameter was specified.

3.2.17 FindInFilesDlg

See also

Properties and Methods

Standard automation properties

[Application](#)

[Parent](#)

[Find](#)

[RegularExpression](#)

[Replace](#)
[DoReplace](#)
[ReplaceOnDisk](#)
[MatchWholeWord](#)
[MatchCase](#)
[SearchLocation](#)
[StartFolder](#)
[IncludeSubfolders](#)
[SearchInProjectFilesDoExternal](#)
[FileExtension](#)
[AdvancedXMLSearch](#)
[XMLElementNames](#)
[XMLElementContents](#)
[XMLAttributeNames](#)
[XMLAttributeContents](#)
[XMLComments](#)
[XMLCDATA](#)
[XMLPI](#)
[XMLRest](#)
[ShowResult](#)

Description

Use this object to configure the search (or replacement) for strings in files. The method [FindInFiles](#) expects a [FindInFilesDlg](#) as parameter.

AdvancedXMLSearch

Property: [AdvancedXMLSearch](#) as Boolean

Description

Specifies if the XML search properties ([XMLElementNames](#), [XMLElementContents](#), [XMLAttributeNames](#), [XMLAttributeContents](#), [XMLComments](#), [XMLCDATA](#), [XMLPI](#) and [XMLRest](#)) are considered. The default is false.

Errors

- 3500 The object is no longer valid.
- 3501 Invalid address for the return parameter was specified.

Application

Property: [Application](#) as [Application](#) (read-only)

Description

Access the XMLSpy application object.

Errors

- 3500 The object is no longer valid.
- 3501 Invalid address for the return parameter was specified.

DoReplace

Property: [DoReplace](#) as Boolean

Description

Specifies if the matched string is replaced by the string defined in [Replace](#). The default is false.

Errors

- 3500 The object is no longer valid.
- 3501 Invalid address for the return parameter was specified.

FileExtension

Property: [FileExtension](#) as String

Description

Specifies the file filter of the files that should be considered during the search. Multiple file filters must be delimited with a semicolon (eg: *.xml;*.dtd;a*.xsd). Use the wildcards * and ? to define the file filter.

Errors

- 3500 The object is no longer valid.
- 3501 Invalid address for the return parameter was specified.

Find

Property: [Find](#) as String

Description

Specifies the string to search for.

Errors

- 3500 The object is no longer valid.
- 3501 Invalid address for the return parameter was specified.

IncludeSubfolders

Property: [IncludeSubfolders](#) as Boolean

Description

Specifies if subfolders are searched too. The default is true.

Errors

- 3500 The object is no longer valid.
- 3501 Invalid address for the return parameter was specified.

MatchCase

Property: [MatchCase](#) as Boolean

Description

Specifies if the search is case sensitive. The default is true.

Errors

- 3500 The object is no longer valid.
- 3501 Invalid address for the return parameter was specified.

MatchWholeWord

Property: [MatchWholeWord](#) as Boolean

Description

Specifies whether the whole word or just a part of it must match. The default is false.

Errors

- 3500 The object is no longer valid.
- 3501 Invalid address for the return parameter was specified.

Parent

Property: [Parent](#) as [Dialogs](#) (read-only)

Description

Access the parent of the object.

Errors

- 3500 The object is no longer valid.
- 3501 Invalid address for the return parameter was specified.

RegularExpression

Property: [RegularExpression](#) as Boolean

Description

Specifies if [Find](#) contains a regular expression. The default is false.

Errors

- 3500 The object is no longer valid.
- 3501 Invalid address for the return parameter was specified.

Replace

Property: [Replace](#) as String

Description

Specifies the replacement string. The matched string is only replaced if [DoReplace](#) is set true.

Errors

- 3500 The object is no longer valid.
- 3501 Invalid address for the return parameter was specified.

ReplaceOnDisk

Property: [ReplaceOnDisk](#) as Boolean

Description

Specifies if the replacement is done directly on disk. The modified file is not opened. The default is false.

Errors

- 3500 The object is no longer valid.
- 3501 Invalid address for the return parameter was specified.

SearchInProjectFilesDoExternal

Property: [SearchInProjectFilesDoExternal](#) as Boolean

Description

Specifies if the external folders in the open project are searched, when a project search is

performed. The default is false.

Errors

- 3500 The object is no longer valid.
- 3501 Invalid address for the return parameter was specified.

SearchLocation

Property: [SearchLocation](#) as [SPYFindInFilesSearchLocation](#)

Description

Specifies the location of the search. The default is spyFindInFiles_Documents.

Errors

- 3500 The object is no longer valid.
- 3501 Invalid address for the return parameter was specified.

ShowResult

Property: [ShowResult](#) as Boolean

Description

Specifies if the result is displayed in the Find in Files output window. The default is false.

Errors

- 3500 The object is no longer valid.
- 3501 Invalid address for the return parameter was specified.

StartFolder

Property: [StartFolder](#) as String

Description

Specifies the folder where the disk search starts.

Errors

- 3500 The object is no longer valid.
- 3501 Invalid address for the return parameter was specified.

XMLAttributeContents

Property: [XMLAttributeContents](#) as Boolean

Description

Specifies if attribute contents are searched when [AdvancedXMLSearch](#) is true. The default is true.

Errors

- 3500 The object is no longer valid.
- 3501 Invalid address for the return parameter was specified.

XMLAttributeNames

Property: [XMLAttributeNames](#) as Boolean

Description

Specifies if attribute names are searched when [AdvancedXMLSearch](#) is true. The default is true.

Errors

- 3500 The object is no longer valid.
- 3501 Invalid address for the return parameter was specified.

XMLCDATA

Property: [XMLCDATA](#) as Boolean

Description

Specifies if CData tags are searched when [AdvancedXMLSearch](#) is true. The default is true.

Errors

- 3500 The object is no longer valid.
- 3501 Invalid address for the return parameter was specified.

XMLComments

Property: [XMLComments](#) as Boolean

Description

Specifies if comments are searched when [AdvancedXMLSearch](#) is true. The default is true.

Errors

- 3500 The object is no longer valid.
- 3501 Invalid address for the return parameter was specified.

XMLElementContents

Property: [XMLElementContents](#) as Boolean

Description

Specifies if element contents are searched when [AdvancedXMLSearch](#) is true. The default is true.

Errors

- 3500 The object is no longer valid.
- 3501 Invalid address for the return parameter was specified.

XMLElementNames

Property: [XMLElementNames](#) as Boolean

Description

Specifies if element names are searched when [AdvancedXMLSearch](#) is true. The default is true.

Errors

- 3500 The object is no longer valid.
- 3501 Invalid address for the return parameter was specified.

XMLPI

Property: [XMLPI](#) as Boolean

Description

Specifies if XML processing instructions are searched when [AdvancedXMLSearch](#) is true. The default is true.

Errors

- 3500 The object is no longer valid.
- 3501 Invalid address for the return parameter was specified.

XMLRest

Property: [XMLRest](#) as Boolean

Description

Specifies if the rest of the XML (which is not covered by the other XML search properties) is searched when [AdvancedXMLSearch](#) is true. The default is true.

Errors

- 3500 The object is no longer valid.
- 3501 Invalid address for the return parameter was specified.

3.2.18 FindInFilesResult

See also

Properties and Methods

Standard automation properties

[Application](#)

[Parent](#)

[Count](#)

[Item](#)

[Path](#)

[Document](#)

Description

This object represents a file that matched the search criteria. It contains a list of [FindInFilesResultMatch](#) objects that describe the matching position.

Application

Property: [Application](#) as [Application](#) (read-only)

Description

Access the XMLSpy application object.

Errors

- 3700 The object is no longer valid.
- 3701 Invalid address for the return parameter was specified.

Count

Property: [Count](#) as long (read-only)

Description

Count of elements in this collection.

Document

Property: [Path](#) as [Document](#) (read-only)

Description

This property returns the [Document](#) object if the matched file is already open in XMLSpy.

Errors

- 3700 The object is no longer valid.
- 3701 Invalid address for the return parameter was specified.

Item

Method: [Item](#)([n](#) as long) as [FindInFilesResultMatch](#)

Description

Gets the element with the index [n](#) from this collection. The first item has index 1.

Parent

Property: [Parent](#) as [FindInFilesResults](#) (read-only)

Description

Access the parent of the object.

Errors

- 3700 The object is no longer valid.
- 3701 Invalid address for the return parameter was specified.

Path

Property: [Path](#) as String (read-only)

Description

Returns the path of the file that matched the search criteria.

Errors

- 3700 The object is no longer valid.
- 3701 Invalid address for the return parameter was specified.

3.2.19 FindInFilesResultMatch

See also

Properties and Methods

Standard automation properties

[Application](#)

[Parent](#)

[Line](#)

[Position](#)

[Length](#)

[LineText](#)

[Replaced](#)

Description

Contains the exact position in the file of the matched string.

Application

Property: [Application](#) as [Application](#) (read-only)

Description

Access the XMLSpy application object.

Errors

- 3800 The object is no longer valid.
- 3801 Invalid address for the return parameter was specified.

Length

Property: [Length](#) as Long (read-only)

Description

Returns the length of the matched string.

Errors

- 3800 The object is no longer valid.
- 3801 Invalid address for the return parameter was specified.

Line

Property: [Line](#) as Long (read-only)

Description

Returns the line number of the match. The line numbering starts with 0.

Errors

- 3800 The object is no longer valid.
- 3801 Invalid address for the return parameter was specified.

LineText

Property: [LineText](#) as String (read-only)

Description

Returns the text of the line.

Errors

- 3800 The object is no longer valid.
- 3801 Invalid address for the return parameter was specified.

Parent

Property: [Parent](#) as [FindInFilesResult](#) (read-only)

Description

Access the parent of the object.

Errors

- 3800 The object is no longer valid.
- 3801 Invalid address for the return parameter was specified.

Position

Property: [Position](#) as Long (read-only)

Description

Returns the start position of the match in the line. The position numbering starts with 0.

Errors

- 3800 The object is no longer valid.
- 3801 Invalid address for the return parameter was specified.

Replaced

Property: [Replaced](#) as Boolean (read-only)

Description

True if the matched string was replaced.

Errors

- 3800 The object is no longer valid.
- 3801 Invalid address for the return parameter was specified.

3.2.20 FindInFilesResults

See also

Properties and Methods

Standard automation properties

[Application](#)

[Parent](#)

[Count](#)

[Item](#)

Description

This is the result of the [FindInFiles](#) method. It is a list of [FindInFilesResult](#) objects.

Application

Property: [Application](#) as [Application](#) (read-only)

Description

Access the XMLSpy application object.

Errors

- 3600 The object is no longer valid.
- 3601 Invalid address for the return parameter was specified.

Count

Property: [Count](#) as long (read-only)

Description

Count of elements in this collection.

Item

Method: [Item](#)([n](#) as long) as [FindInFilesResult](#)

Description

Gets the element with the index [n](#) from this collection. The first item has index 1.

Parent

Property: [Parent](#) as [Application](#) (read-only)

Description

Access the parent of the object.

Errors

- 3600 The object is no longer valid.
- 3601 Invalid address for the return parameter was specified.

3.2.21 GenerateSampleXMLDlg

See also

Properties and Methods

Standard automation properties

[Application](#)
[Parent](#)

[NonMandatoryAttributes](#)
[RepeatCount](#)
[FillAttributesWithSampleData](#)
[FillElementsWithSampleData](#)
[ContentOfNillableElementsIsNonMandatory](#)
[TryToUseNonAbstractTypes](#)

[Optimization](#)
[SchemaOrDTDAssignment](#)
[LocalNameOfRootElement](#)
[NamespaceURIOfRootElement](#)
[OptionsDialogAction](#)

Properties that are no longer supported

[NonMandatoryElements - obsolete](#)
[TakeFirstChoice - obsolete](#)
[FillWithSampleData - obsolete](#)

Description

Used to set the parameters for the generation of sample XML instances based on a W3C schema or DTD.

Application

Property: [Application](#) as [Application](#) (read-only)

Description

Access the XMLSpy application object.

Errors

- 2200 The object is no longer valid.
- 2201 Invalid address for the return parameter was specified.

Parent

Property: [Parent](#) as [Dialogs](#) (read-only)

Description

Access the parent of the object.

Errors

- 2200 The object is no longer valid.
- 2201 Invalid address for the return parameter was specified.

NonMandatoryAttributes

Property: [NonMandatoryAttributes](#) as Boolean

Description

If true attributes which are not mandatory are created in the sample XML instance file.

Errors

- 2200 The object is no longer valid.
- 2201 Invalid address for the return parameter was specified.

NonMandatoryElements - obsolete

Property: [NonMandatoryElements](#) as Boolean

Description

Do no longer use this property. Use [Optimization](#), instead.

Errors

0001 The property is no longer accessible.

TakeFirstChoice - obsolete

Property: [TakeFirstChoice](#) as Boolean

Description

Do no longer use this property.

Errors

0001 The property is no longer accessible.

RepeatCount

Property: [RepeatCount](#) as long

Description

Number of elements to create for repeated types.

Errors

2200 The object is no longer valid.

2201 Invalid address for the return parameter was specified.

FillWithSampleData - obsolete

Property: [FillWithSampleData](#) as Boolean

Description

Do no longer access this property. Use [FillAttributesWithSampleData](#) and [FillElementsWithSampleData](#), instead.

Errors

0001 The property is no longer accessible.

FillElementsWithSampleData

Property: [FillElementsWithSampleData](#) as Boolean

Description

If true, elements will have sample content.

Errors

2200 The object is no longer valid.

2201 Invalid address for the return parameter was specified.

FillAttributesWithSampleData

Property: [FillAttributesWithSampleData](#) as Boolean

Description

If true, attributes will have sample content.

Errors

2200 The object is no longer valid.

2201 Invalid address for the return parameter was specified.

ContentOfNillableElementsIsNonMandatory

Property: [ContentOfNillableElementsIsNonMandatory](#) as Boolean

Description

If true, the contents of elements that are nillable will not be treated as mandatory.

Errors

- 2200 The object is no longer valid.
- 2201 Invalid address for the return parameter was specified.

TryToUseNonAbstractTypes

Property: [TryToUseNonAbstractTypes](#) as Boolean

Description

If true, tries to use a non-abstract type for xsi:type, if element has an abstract type.

Errors

- 2200 The object is no longer valid.
- 2201 Invalid address for the return parameter was specified.

Optimization

Property: [Optimization](#) as [SPYSampleXMLGenerationOptimization](#)

Description

Specifies which elements will be generated.

Errors

- 2200 The object is no longer valid.
- 2201 Invalid address for the return parameter was specified.

SchemaOrDTDAssignment

Property: [SchemaOrDTDAssignment](#) as [SPYSampleXMLGenerationSchemaOrDTDAssignment](#)

Description

Specifies in which way a reference to the related schema or DTD - which is this document - will be generated into the sample XML.

Errors

- 2200 The object is no longer valid.
- 2201 Invalid address for the return parameter was specified.

LocalNameOfRootElement

Property: [LocalNameOfRootElement](#) as String

Description

Specifies the local name of the root element for the generated sample XML.

Errors

- 2200 The object is no longer valid.
- 2201 Invalid address for the return parameter was specified.

NamespaceURIOfRootElement

Property: [NamespaceURIOfRootElement](#) as String

Description

Specifies the namespace URI of the root element for the generated sample XML.

Errors

- 2200 The object is no longer valid.
- 2201 Invalid address for the return parameter was specified.

OptionsDialogAction

Property: [OptionsDialogAction](#) as [SPYDAbgAction](#)

Description

To allow your script to fill in the default values and let the user see and react on the dialog, set this property to the value *spyDialogUserInput(2)*. If you want your script to define all the options in the schema documentation dialog without any user interaction necessary, use *spyDialogOK(0)*. Default is *spyDialogOK*.

Errors

- 2200 The object is no longer valid.
- 2201 Invalid value has been used to set the property.
Invalid address for the return parameter was specified.

3.2.22 GridView

See also

Methods

[Deselect](#)
[Select](#)

[SetFocus](#)

Properties

[CurrentFocus](#)
[IsVisible](#)

Description

GridView Class

Events

OnBeforeDrag

See also

Event: [OnBeforeDrag\(\)](#) as Boolean

XMLSpy scripting environment - VBScript:

```
Function On_BeforeDrag()
    'On_BeforeStartEditing=False 'bprohibitdragging
EndFunction
```

XMLSpy scripting environment - JScript:

```
function On_BeforeDrag()
{
    //return false; 'bprohibitdragging*/
}
```

XMLSpy IDE Plugin:

```
IXMLSpyPlugin.OnEvent (4, ...) //nEventId=4
```

Description

This event gets fired on an attempt to drag an XMLData element on the grid view. Return *false* to prevent dragging the data element to a different position.

OnBeforeDrop**See also**

Event: OnBeforeDrop(*objXMLData* as [XMLData](#)) as Boolean

XMLSpy scripting environment - VBScript:

```
Function On_BeforeDrop(objXMLData)
    'On_BeforeStartEditing=False 'bprohibitdragging
EndFunction
```

XMLSpy scripting environment - JScript:

```
function On_BeforeDrop(objXMLData)
{
    //return false; 'bprohibitdragging*/
}
```

XMLSpy IDE Plugin:

```
IXMLSpyPlugin.OnEvent (5, ...) //nEventId=5
```

Description

This event gets fired on an attempt to drop a previously dragged XMLData element on the grid view. Return *false* to prevent the data element to be moved from its original position to the drop destination position.

OnBeforeStartEditing**See also**

Event: OnBeforeStartEditing(*objXMLData* as [XMLData](#), *bEditingName* as Boolean) as Boolean

XMLSpy scripting environment - VBScript:

```
Function On_BeforeStartEditing(objXMLData, bEditingName)
    'On_BeforeStartEditing=False 'bprohibitdragging
EndFunction
```

```
EndFunction
```

XMLSpy scripting environment - JScript:

```
function On_BeforeStartEditing(objXMLData, bEditingName)
{
    //unfals: "onBeforeStartEditing"
}
```

XMLSpy IDE Plugin:

```
IXMLSpyPlugin.OnEvent (1, ...) //nEventId=1
```

Description

This event gets fired before the editing mode for a grid cell gets entered. If the parameter *bEditingName* is true, the name part of the element will be edited, if its value is false, the value part will be edited.

OnEditingFinished

See also

Event: *OnEditingFinished*(*objXMLData* as [XMLData](#), *bEditingName* as Boolean)

XMLSpy scripting environment - VBScript:

```
Function On_EditingFinished(objXMLData, bEditingName)
EndFunction
```

XMLSpy scripting environment - JScript:

```
function On_EditingFinished(objXMLData, bEditingName)
{
}
```

XMLSpy IDE Plugin:

```
IXMLSpyPlugin.OnEvent (2, ...) //nEventId=2
```

Description

This event gets fired when the editing mode of a grid cell gets left. The parameter *bEditingName* specifies if the name part of the element has been edited.

OnFocusChanged

See also

Event: *OnFocusChanged*(*objXMLData* as [XMLData](#), *bSetFocus* as Boolean, *bEditingName* as Boolean)

XMLSpy scripting environment - VBScript:

```
Function On_FocusChanged(objXMLData, bSetFocus, bEditingName)
EndFunction
```

XMLSpy scripting environment - JScript:

```
function On_FocusChanged(objXMLData, bSetFocus, bEditingName)
{
}
```

XMLSpy IDE Plugin:

`IXMLSpyPlugIn.OnEvent (3, ...) //nEvent=3`

Description

This event gets fired whenever a grid cell receives or loses the cursor focus. If the parameter *bEditingName* is *true*, focus of the name part of the grid element has changed. Otherwise, focus of the value part has changed.

CurrentFocus**See also**

Property: `CurrentFocus` as [XMLData](#)

Description

Holds the XML element with the current focus. This property is read-only.

Deselect**See also**

Method: `Deselect(pData as XMLData)`

Description

Deselects the element `pData` in the grid view.

IsVisible**See also**

Property: `IsVisible` as Boolean

Description

True if the grid view is the active view of the document. This property is read-only.

Select**See also**

Method: `Select (pData as XMLData)`

Description

Selects the XML element `pData` in the grid view.

SetFocus**See also**

Method: `SetFocus (pFocusData as XMLData)`

Description

Sets the focus to the element `pFocusData` in the grid view.

3.2.23 SchemaDocumentationDlg

See also

Properties and Methods

Standard automation properties

[Application](#)

[Parent](#)

Interaction and visibility properties

[OutputFile](#)

[OutputFileDialogAction](#)

[OptionsDialogAction](#)

[ShowProgressbar](#)

[ShowResult](#)

Document generation options and methods

[OutputFormat](#)

[UseFixedDesign](#)

[SPSFile](#)

[EmbedDiagrams](#)

[DiagramFormat](#)

[MultipleOutputFiles](#)

[EmbedCSSInHTML](#)

[CreateDiagramsFolder](#)

[GenerateRelativeLinks](#)

[IncludeAll](#)

[IncludeIndex](#)

[IncludeGlobalAttributes](#)

[IncludeGlobalElements](#)

[IncludeLocalAttributes](#)

[IncludeLocalElements](#)

[IncludeGroups](#)

[IncludeComplexTypes](#)

[IncludeSimpleTypes](#)

[IncludeAttributeGroups](#)

[IncludeRedefines](#)

[IncludeReferencedSchemas](#)

[AllDetails](#)

[ShowDiagram](#)

[ShowNamespace](#)

[ShowType](#)

[ShowChildren](#)

[ShowUsedBy](#)

[ShowProperties](#)

[ShowSingleFacets](#)

[ShowPatterns](#)

[ShowEnumerations](#)

[ShowAttributes](#)

[ShowIdentityConstraints](#)

[ShowAnnotations](#)

[ShowSourceCode](#)

Description

This object combines all options for schema document generation as they are available through

user interface dialog boxes in XMLSpy. The document generation options are initialized with the values used during the last generation of schema documentation. However, before using the object you have to set the [SetOutputFile](#) property to a valid file path. Use [OptionsDialogAction](#), [OutputFileDialogAction](#) and [ShowProgressBar](#) to specify the level of user interaction desired. You can use [HideAll](#) and [ShowAll](#) to set whole option groups at once or the individual properties to operate on a finer granularity.

AllDetails

See also

Method: [AllDetails](#) ([i_bDetailsOn](#) as Boolean)

Description

Use this method to turn all details options on or off.

Errors

2900 The object is no longer valid.

Application

See also

Property: [Application](#) as [Application](#) (read-only)

Description

Access the XMLSpy application object.

Errors

2900 The object is no longer valid.

2901 Invalid address for the return parameter was specified.

CreateDiagramsFolder

See also

Property: [CreateDiagramsFolder](#) as Boolean

Description

Set this property to `true`, to create a directory for the created images. Otherwise the diagrams will be created next to the documentation. This property is only available when the diagrams are not embedded. The property is initialized with the value used during the last call to [DocumentGenerateWSDLDocumentation](#). The default for the first run is false.

Errors

2900 The object is no longer valid.

2901 Invalid address for the return parameter was specified.

DiagramFormat

See also

Property: [DiagramFormat](#) as [SPYImageKind](#)

Description

This property specifies the generated diagram image type. This property is not available for HTML documentation. The property is initialized with the value used during the last call to [DocumentGenerateSchemaDocumentation](#). The default for the first run is PNG.

Errors

- 2900 The object is no longer valid.
- 2901 Invalid address for the return parameter was specified.

EmbedCSSInHTML

See also

Property: [EmbedCSSInHTML](#) as [Boolean](#)

Description

Set this property to `true`, to embed the CSS data in the generated HTML document. Otherwise a separate file will be created and linked. This property is only available for HTML documentation. The property is initialized with the value used during the last call to [DocumentGenerateWSDLDocumentation](#). The default for the first run is true.

Errors

- 2900 The object is no longer valid.
- 2901 Invalid address for the return parameter was specified.

EmbedDiagrams

See also

Property: [EmbedDiagrams](#) as [Boolean](#)

Description

Set this property to `true`, to embed the diagrams in the generated document. This property is not available for HTML documentation. The property is initialized with the value used during the last call to [DocumentGenerateSchemaDocumentation](#). The default for the first run is true.

Errors

- 2900 The object is no longer valid.
- 2901 Invalid address for the return parameter was specified.

GenerateRelativeLinks

See also

Property: [GenerateRelativeLinks](#) as [Boolean](#)

Description

Set this property to `true`, to create relative paths to local files. This property is not available for HTML documentation. The property is initialized with the value used during the last call to [DocumentGenerateSchemaDocumentation](#). The default for the first run is false.

Errors

- 2900 The object is no longer valid.
- 2901 Invalid address for the return parameter was specified.

IncludeAll**See also**

Method: [IncludeAll](#) (`i_bInclude` as Boolean)

Description

Use this method to mark or unmark all include options.

Errors

- 2900 The object is no longer valid.

IncludeAttributeGroups**See also**

Property: [IncludeAttributeGroups](#) as Boolean

Description

Set this property to `true`, to include attribute groups in the schema documentation. The property is initialized with the value used during the last call to [DocumentGenerateSchemaDocumentation](#). The default for the first run is true.

Errors

- 2900 The object is no longer valid.
- 2901 Invalid address for the return parameter was specified.

IncludeComplexTypes**See also**

Property: [IncludeComplexTypes](#) as Boolean

Description

Set this property to `true`, to include complex types in the schema documentation. The property is initialized with the value used during the last call to [DocumentGenerateSchemaDocumentation](#). The default for the first run is true.

Errors

- 2900 The object is no longer valid.
- 2901 Invalid address for the return parameter was specified.

IncludeGlobalAttributes

See also

Property: [IncludeGlobalAttributes](#) as Boolean

Description

Set this property to `true`, to include global attributes in the schema documentation. The property is initialized with the value used during the last call to [DocumentGenerateSchemaDocumentation](#). The default for the first run is true.

Errors

- 2900 The object is no longer valid.
- 2901 Invalid address for the return parameter was specified.

IncludeGlobalElements

See also

Property: [IncludeGlobalElements](#) as Boolean

Description

Set this property to `true`, to include global elements in the schema documentation. The property is initialized with the value used during the last call to [DocumentGenerateSchemaDocumentation](#). The default for the first run is true.

Errors

- 2900 The object is no longer valid.
- 2901 Invalid address for the return parameter was specified.

IncludeGroups

See also

Property: [IncludeGroups](#) as Boolean

Description

Set this property to `true`, to include groups in the schema documentation. The property is initialized with the value used during the last call to [DocumentGenerateSchemaDocumentation](#). The default for the first run is true.

Errors

- 2900 The object is no longer valid.
- 2901 Invalid address for the return parameter was specified.

IncludeIndex

See also

Property: [IncludeIndex](#) as Boolean

Description

Set this property to `true`, to include an index in the schema documentation. The property is initialized with the value used during the last call to [DocumentGenerateSchemaDocumentation](#).

The default for the first run is true.

Errors

- 2900 The object is no longer valid.
- 2901 Invalid address for the return parameter was specified.

IncludeLocalAttributes**See also**

Property: [IncludeLocalAttributes](#) as Boolean

Description

Set this property to `true`, to include local attributes in the schema documentation. The property is initialized with the value used during the last call to [DocumentGenerateSchemaDocumentation](#). The default for the first run is true.

Errors

- 2900 The object is no longer valid.
- 2901 Invalid address for the return parameter was specified.

IncludeLocalElements**See also**

Property: [IncludeLocalElements](#) as Boolean

Description

Set this property to `true`, to include local elements in the schema documentation. The property is initialized with the value used during the last call to [DocumentGenerateSchemaDocumentation](#). The default for the first run is true.

Errors

- 2900 The object is no longer valid.
- 2901 Invalid address for the return parameter was specified.

IncludeRedefines**See also**

Property: [IncludeRedefines](#) as Boolean

Description

Set this property to `true`, to include redefines in the schema documentation. The property is initialized with the value used during the last call to [DocumentGenerateSchemaDocumentation](#). The default for the first run is true.

Errors

- 2900 The object is no longer valid.
- 2901 Invalid address for the return parameter was specified.

IncludeReferencedSchemas**See also**

Property: [IncludeReferencedSchemas](#) as Boolean

Description

Set this property to `true`, to include referenced schemas in the schema documentation. The property is initialized with the value used during the last call to [DocumentGenerateSchemaDocumentation](#). The default for the first run is true.

Errors

- 2900 The object is no longer valid.
- 2901 Invalid address for the return parameter was specified.

IncludeSimpleTypes

See also

Property: [IncludeSimpleTypes](#) as Boolean

Description

Set this property to `true`, to include simple types in the schema documentation. The property is initialized with the value used during the last call to [DocumentGenerateSchemaDocumentation](#). The default for the first run is true.

Errors

- 2900 The object is no longer valid.
- 2901 Invalid address for the return parameter was specified.

MultipleOutputFiles

See also

Property: [MultipleOutputFiles](#) as Boolean

Description

Set this property to `true`, to split the documentation files. The property is initialized with the value used during the last call to [DocumentGenerateSchemaDocumentation](#). The default for the first run is false.

Errors

- 2900 The object is no longer valid.
- 2901 Invalid value has been used to set the property.
Invalid address for the return parameter was specified.

OptionsDialogAction

See also

Property: [OptionsDialogAction](#) as [SPYDialogAction](#)

Description

To allow your script to fill in the default values and let the user see and react on the dialog, set this property to the value `spyDialogUserInput(2)`. If you want your script to define all the options in the schema documentation dialog without any user interaction necessary, use `spyDialogOK(0)`. Default is `spyDialogOK`.

Errors

- 2900 The object is no longer valid.
- 2901 Invalid value has been used to set the property.
Invalid address for the return parameter was specified.

OutputFile

See also

Property: [OutputFile](#) as [String](#)

Description

Full path and name of the file that will contain the generated documentation. In case of HTML output, additional '.png' files will be generated based on this filename. The default value for this property is an empty string and needs to be replaced before using this object in a call to [DocumentGenerateSchem aDocumentation](#).

Errors

- 2900 The object is no longer valid.
- 2901 Invalid address for the return parameter was specified.

OutputFileDialogAction

See also

Property: [OutputFileDialogAction](#) as [SPYDialogAction](#)

Description

To allow the user to select the output file with a file selection dialog, set this property to [spyDialogUserInput\(2\)](#). If the value stored in [OutputFile](#) should be taken and no user interaction should occur, use [spyDialogOK\(0\)](#). Default is [spyDialogOK](#).

Errors

- 2900 The object is no longer valid.
- 2901 Invalid value has been used to set the property.
Invalid address for the return parameter was specified.

OutputFormat

See also

Property: [OutputFormat](#) as [SPYSchem aDocumentationForm at](#)

Description

Defines the kind of documentation that will be generated: HTML (value=0), MS-Word (value=1), or RTF (value=2). The property gets initialized with the value used during the last call to [DocumentGenerateSchem aDocumentation](#). The default for the first run is HTML.

Errors

- 2900 The object is no longer valid.
- 2901 Invalid value has been used to set the property.
Invalid address for the return parameter was specified.

Parent**See also**

Property: [Parent](#) as [Dialogs](#) (read-only)

Description

Access the parent of the object.

Errors

- 2900 The object is no longer valid.
- 2901 Invalid address for the return parameter was specified.

ShowAnnotations**See also**

Property: [ShowAnnotations](#) as [Boolean](#)

Description

Set this property to `true`, to show the annotations to a type definition in the schema documentation. The property is initialized with the value used during the last call to [DocumentGenerateSchemaDocumentation](#). The default for the first run is true.

Errors

- 2900 The object is no longer valid.
- 2901 Invalid address for the return parameter was specified.

ShowAttributes**See also**

Property: [ShowAttributes](#) as [Boolean](#)

Description

Set this property to `true`, to show the type definitions attributes in the schema documentation. The property is initialized with the value used during the last call to [DocumentGenerateSchemaDocumentation](#). The default for the first run is true.

Errors

- 2900 The object is no longer valid.
- 2901 Invalid address for the return parameter was specified.

ShowChildren**See also**

Property: [ShowChildren](#) as [Boolean](#)

Description

Set this property to `true`, to show the children of a type definition as links in the schema documentation. The property is initialized with the value used during the last call to [DocumentGenerateSchemaDocumentation](#). The default for the first run is true.

Errors

- 2900 The object is no longer valid.
- 2901 Invalid address for the return parameter was specified.

ShowDiagram**See also**

Property: [ShowDiagram](#) as Boolean

Description

Set this property to `true`, to show type definitions as diagrams in the schema documentation. The property is initialized with the value used during the last call to [DocumentGenerateSchemaDocumentation](#). The default for the first run is true.

Errors

- 2900 The object is no longer valid.
- 2901 Invalid address for the return parameter was specified.

ShowEnumerations**See also**

Property: [ShowEnumerations](#) as Boolean

Description

Set this property to `true`, to show the enumerations contained in a type definition in the schema documentation. The property is initialized with the value used during the last call to [DocumentGenerateSchemaDocumentation](#). The default for the first run is true.

Errors

- 2900 The object is no longer valid.
- 2901 Invalid address for the return parameter was specified.

ShowIdentityConstraints**See also**

Property: [ShowIdentityConstraints](#) as Boolean

Description

Set this property to `true`, to show a type definitions identity constraints in the schema documentation. The property is initialized with the value used during the last call to [DocumentGenerateSchemaDocumentation](#). The default for the first run is true.

Errors

- 2900 The object is no longer valid.
- 2901 Invalid address for the return parameter was specified.

ShowNamespace

See also

Property: [ShowNamespace](#) as Boolean

Description

Set this property to `true`, to show the namespace of type definitions in the schema documentation. The property is initialized with the value used during the last call to [DocumentGenerateSchemaDocumentation](#). The default for the first run is true.

Errors

- 2900 The object is no longer valid.
- 2901 Invalid address for the return parameter was specified.

ShowPatterns

See also

Property: [ShowPatterns](#) as Boolean

Description

Set this property to `true`, to show the patterns of a type definition in the schema documentation. The property is initialized with the value used during the last call to [DocumentGenerateSchemaDocumentation](#). The default for the first run is true.

Errors

- 2900 The object is no longer valid.
- 2901 Invalid address for the return parameter was specified.

ShowProgressBar

See also

Property: [ShowProgressBar](#) as Boolean

Description

Set this property to `true`, to make the window showing the document generation progress visible. Use `false`, to hide it. Default is `false`.

Errors

- 2900 The object is no longer valid.
- 2901 Invalid address for the return parameter was specified.

ShowProperties

See also

Property: [ShowProperties](#) as Boolean

Description

Set this property to `true`, to show the type definition properties in the schema documentation. The property is initialized with the value used during the last call to

[DocumentGenerateSchemaDocumentation](#). The default for the first run is true.

Errors

- 2900 The object is no longer valid.
- 2901 Invalid address for the return parameter was specified.

ShowResult**See also**

Property: [ShowResult](#) as Boolean

Description

Set this property to `true`, to automatically open the resulting document when generation was successful. HTML documentation will be opened in XMLSpy. To show Word documentation, MS-Word will be started. The property gets initialized with the value used during the last call to [DocumentGenerateSchemaDocumentation](#). The default for the first run is true.

Errors

- 2900 The object is no longer valid.
- 2901 Invalid address for the return parameter was specified.

ShowSingleFacets**See also**

Property: [ShowSingleFacets](#) as Boolean

Description

Set this property to `true`, to show the facets of a type definition in the schema documentation. The property is initialized with the value used during the last call to [DocumentGenerateSchemaDocumentation](#). The default for the first run is true.

Errors

- 2900 The object is no longer valid.
- 2901 Invalid address for the return parameter was specified.

ShowSourceCode**See also**

Property: [ShowSourceCode](#) as Boolean

Description

Set this property to `true`, to show the XML source code for type definitions in the schema documentation. The property is initialized with the value used during the last call to [DocumentGenerateSchemaDocumentation](#). The default for the first run is true.

Errors

- 2900 The object is no longer valid.
- 2901 Invalid address for the return parameter was specified.

ShowType

See also

Property: [ShowType](#) as Boolean

Description

Set this property to `true`, to show the type of type definitions in the schema documentation. The property is initialized with the value used during the last call to [DocumentGenerateSchemaDocumentation](#). The default for the first run is true.

Errors

- 2900 The object is no longer valid.
- 2901 Invalid address for the return parameter was specified.

ShowUsedBy

See also

Property: [ShowUsedBy](#) as Boolean

Description

Set this property to `true`, to show the used-by relation for type definitions in the schema documentation. The property is initialized with the value used during the last call to [DocumentGenerateSchemaDocumentation](#). The default for the first run is true.

Errors

- 2900 The object is no longer valid.
- 2901 Invalid address for the return parameter was specified.

SPSFile

See also

Property: [SPSFile](#) as String

Description

Full path and name of the SPS file that will be used to generate the documentation.

Errors

- 2900 The object is no longer valid.
- 2901 Invalid address for the return parameter was specified.

UseFixedDesign

See also

Property: [UseFixedDesign](#) as Boolean

Description

Specifies whether the documentation should be created with a fixed design or with a design specified by a SPS file (which requires StyleVision).

Errors

- 2900 The object is no longer valid.
- 2901 Invalid address for the return parameter was specified.

3.2.24 SpyProject**See also****Methods**

[CloseProject](#)
[SaveProject](#)
[SaveProjectAs](#)

Properties

[RootItems](#)
[ProjectFile](#)

Description

~~SpyProject~~ Class

CloseProject**See also**

Declaration: [CloseProject](#)(*bDiscardChanges* as Boolean, *bCloseFiles* as Boolean, *bDialog* as Boolean)

Parameters

bDiscardChanges

Set *bDiscardChanges* to FALSE if you want to save the changes of the open project files and the project.

bCloseFiles

Set *bCloseFiles* to TRUE to close all open project files.

bDialog

Show dialogs for user input.

Description

[CloseProject](#) closes the current project.

ProjectFile**See also**

Declaration: [ProjectFile](#) as String

Description

Path and filename of the project.

RootItems

See also

Declaration: [RootItems](#) as [SpyProjectItems](#)

Description

Root level of collection of project items.

SaveProject

See also

Declaration: [SaveProject](#)

Description

[SaveProject](#) saves the current project.

SaveProjectAs

See also

Declaration: [SaveProjectAs](#) ([strPath](#) as String, [bDialog](#) as Boolean)

Parameters

[strPath](#)

Full path with file name of new project file.

[bDialog](#)

If [bDialog](#) is TRUE, a file-dialog will be displayed.

Description

[SaveProjectAs](#) stores the project data into a new location.

3.2.25 SpyProjectItem

See also

Methods

[Open](#)

Properties

[ChildItems](#)

[ParentItem](#)

[FileExtensions](#)

[ItemType](#)

[Name](#)

[Path](#)

[ValidateWith](#)

[XMLForXSLTransformation](#)

[XSLForXMLTransformation](#)

[XSLTransformationFileExtension](#)

[XSLTransformationFolder](#)

Description

~~SpyProject~~ Class

ChildItems

See also

Declaration: [ChildItems](#) as [SpyProjectItems](#)

Description

If the item is a folder, [ChildItems](#) is the collection of the folder content.

FileExtensions

See also

Declaration: [FileExtensions](#) as String

Description

Used to set the file extensions if the project item is a folder.

ItemType

See also

Declaration: [ItemType](#) as [SPYProjectItemTypes](#)

Description

This property is read-only.

Name

See also

Declaration: [Name](#) as String

Description

Name of the project item. This property is read-only.

Open

See also

Declaration: [Open](#) as [Document](#)

Return Value

The project item opened as document.

Description

Opens the project item.

ParentItem

See also

Declaration: `ParentItem` as [SpyProjectItem](#)

Description

Parent item of the current project item. Can be NULL (Nothing) if the project item is a top-level item.

Path

See also

Declaration: `Path` as String

Description

Path of project item. This property is read-only.

ValidateWith

See also

Declaration: `ValidateWith` as String

Description

Used to set the schema/DTD for validation.

XMLForXSLTransformation

See also

Declaration: `XMLForXSLTransformation` as String

Description

Used to set the XML for XSL transformation.

XSLForXMLTransformation

See also

Declaration: `XSLForXMLTransformation` as String

Description

Used to set the XSL for XML transformation.

XSLTransformationFileExtension

See also

Declaration: `XSLTransformationFileExtension` as String

Description

Used to set the file extension for XSL transformation output files.

XSLTransformationFolder

See also

Declaration: [XSLTransformationFolder](#) as String

Description

Used to set the destination folder for XSL transformation output files.

3.2.26 SpyProjectItems

See also

Methods

[AddFile](#)

[AddFolder](#)

[AddURL](#)

[RemoveItem](#)

Properties

[Count](#)

[Item](#)

Description

[SpyProjectItems](#) Class

AddFile

See also

Declaration: [AddFile](#) ([strPath](#) as String)

Parameters

[strPath](#)

Full path with file name of new project item

Description

The method adds a new file to the collection of project items.

AddFolder

See also

Declaration: [AddFolder](#) ([strName](#) as String)

Parameters

[strName](#)

Name of the new folder.

Description

The method [AddFolder](#) adds a folder with the name [strName](#) to the collection of project items.

AddURL

See also

Declaration: `AddURL` (*strURL* as String, *nURLType* as [SPYURLTypes](#), *strUser* as String, *strPassword* as String, *bSave* as Boolean)

Description

strURL

URL to open as document.

nURLType

Type of document to open. Set to -1 for auto detection.

strUser

Name of the user if required. Can be empty.

strPassword

Password for authentication. Can be empty.

bSave

Save user and password information.

Description

The method adds an URL item to the project collection.

Count

See also

Declaration: `Count` as long

Description

This property gets the count of project items in the collection. The property is read-only.

Item

See also

Declaration: `Item` (*n* as long) as [SpyProjectItem](#)

Description

Retrieves the *n*-th element of the collection of project items. The first item has index 1.

RemoveItem

See also

Declaration: `RemoveItem` (*pItem* as [SpyProjectItem](#))

Description

`RemoveItem` deletes the item *pItem* from the collection of project items.

3.2.27 TextImportExportSettings

See also

Properties for import only

[ImportFile](#)

Properties for export only

[DestinationFolder](#)

[FileExtension](#)

[CommentIncluded](#)

[RemoveDelimiter](#)

[RemoveNewline](#)

Properties for import and export

[HeaderRow](#)

[FieldDelimiter](#)

[EnclosingCharacter](#)

[Encoding](#)

[EncodingByteOrder](#)

Description

`TextImportExportSettings` contains options common to text import and export functions.

CommentIncluded

See also

Property: `CommentIncluded` as Boolean

Description

This property tells whether additional comments are added to the generated text file. Default is true. This property is used only when exporting to text files.

DestinationFolder

See also

Property: `DestinationFolder` as String

Description

The property `DestinationFolder` sets the folder where the created files are saved during text export.

EnclosingCharacter

See also

Property: `EnclosingCharacter` as [SPYTextEnclosing](#)

Description

This property defines the character that encloses all field values for import and export. Default is [spyNoEnclosing](#).

Encoding

See also

Property: [Encoding](#) as String

Description

The property `Encoding` sets the character encoding for the text files for importing and exporting.

EncodingByteOrder

See also

Property: [EncodingByteOrder](#) as [SPYEncodingByteOrder](#)

Description

The property `EncodingByteOrder` sets the byte order for Unicode characters. Default is [spyNONE](#).

FieldDelimiter

See also

Property: [FieldDelimiter](#) as [SPYTextDelimiters](#)

Description

The property `FieldDelimiter` defines the delimiter between the fields during import and export. Default is [spyTabulator](#).

FileExtension

See also

Property: [FileExtension](#) as String

Description

This property sets the file extension for files created on text export.

HeaderRow

See also

Property: [HeaderRow](#) as Boolean

Description

The property `HeaderRow` is used during import and export. Set `HeaderRow` true on import, if the first line of the text file contains the names of the columns. Set `HeaderRow` true on export, if the first line in the created text files should contain the name of the columns. Default value is true.

ImportFile**See also****Property:** [ImportFile](#) as String**Description**

This property is used to set the text file for import. The string has to be a full qualified path. See also [Import and Export](#).

RemoveDelimiter**See also****Property:** [RemoveDelimiter](#) as Boolean**Description**

The property [RemoveDelimiter](#) defines whether characters in the text that are equal to the delimiter character are removed. Default is false. This property is used only when exporting to text files.

RemoveNewline**See also****Property:** [RemoveNewline](#) as Boolean**Description**

The property [RemoveNewline](#) defines whether newline characters in the text are removed. Default is false. This property is used only when exporting to text files.

3.2.28 TextView**See also****Properties and Methods**[Application](#)
[Parent](#)[LineFromPosition](#)
[PositionFromLine](#)
[LineLength](#)
[SelText](#)
[GetRangeText](#)
[ReplaceText](#)
[MoveCaret](#)
[GoToLineChar](#)
[SelectText](#)
[SelectionStart](#)
[SelectionEnd](#)
[Text](#)
[LineCount](#)
[Length](#)

Description**Events*****OnBeforeShowSuggestions*****See also**

Event: [OnBeforeShowSuggestions\(\)](#) as Boolean

Description

This event gets fired before a suggestion window is shown. The [Document](#) property [Suggestions](#) contains a string array that is recommended to the user. It is possible to modify the displayed recommendations during this event. Before doing so you have to assign an empty array to the [Suggestions](#) property. The best location for this is the [OnDocumentOpened](#) event. To prevent the suggestion window to show up return [false](#) and [true](#) to continue its display.

Examples

Given below are examples of how this event can be scripted.

XMLSpy scripting environment - VBScript:

```
Function On\_BeforeShowSuggestions\(\)
EndFunction
```

XMLSpy scripting environment - JScript:

```
function On\_BeforeShowSuggestions\(\)
{
}
```

XMLSpy IDE Plugin:

```
IXMLSpyPlugin.OnEvent (33, ...) //nEventId=33
```

OnChar**See also**

Event: [OnChar](#)([nChar](#) as Long, [bExistSuggestions](#) as Boolean) as Boolean

Description

This event gets fired on each key stroke. The parameter [nChar](#) is the key that was pressed and [bExistSuggestions](#) tells whether a XMLSpy generated suggestions window is displayed after this key. The [Document](#) property [Suggestions](#) contains a string array that is recommended to the user. It is possible to modify the displayed recommendations during this event. Before doing so you have to assign an empty array to the [Suggestions](#) property. The best location for this is the [OnDocumentOpened](#) event. To prevent the suggestion window to show up return [false](#) and [true](#) to continue its display.

It is also possible to create a new suggestions window when none is provided by XMLSpy. Set the [Document](#) property [Suggestions](#) to a string array with your recommendations and return [true](#).

This event is fired before the [OnBeforeShowSuggestions](#) event. If you prevent to show the suggestion window by returning [false](#) then [OnBeforeShowSuggestions](#) is not fired.

Examples

Given below are examples of how this event can be scripted.

XMLSpy scripting environment - VBScript:

```
Function On_Char(nChar, bExistsSuggestions )
EndFunction
```

XMLSpy scripting environment - JScript:

```
function On_Char(nChar, bExistsSuggestions )
{
}
}
```

XMLSpy IDE Plugin:

```
IXMLSpyPlugin.OnEvent (35, ...) //nEventId=35
```

Application

Property: [Application](#) as [Application](#) (read-only)

Description

Access the XMLSpy application object.

Errors

- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

GetRangeText

Method: [GetRangeText](#)([nStart](#) as Long, [nEnd](#) as Long) as String

Description

Returns the text in the specified range.

Errors

- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

GoToLineChar

Method: [GoToLineChar](#)([nLine](#) as Long, [nChar](#) as Long)

Description

Moves the caret to the specified line and character position.

Errors

- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

Length

Property: [Length](#) as Long

Description

Returns the character count of the document.

Errors

- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

LineCount

Property: [LineCount](#) as Long

Description

Returns the number of lines in the document.

Errors

- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

LineFromPosition

Method: [LineFromPosition](#)([nCharPos](#) as Long) as Long

Description

Returns the line number of the character position.

Errors

- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

LineLength

Method: [LineLength](#)([nLine](#) as Long) as Long

Description

Returns the length of the line.

Errors

- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

MoveCaret

Method: [MoveCaret](#)([nDiff](#) as Long)

Description

Moves the caret [nDiff](#) characters.

Errors

- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

Parent

Property: [Parent](#) as [Document](#) (read-only)

Description

Access the parent of the object.

Errors

- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

PositionFromLine

Method: `PositionFromLine`(`nLine` as Long) as Long

Description

Returns the start position of the line.

Errors

- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

ReplaceText

Method: `ReplaceText`(`nPosFrom` as Long, `nPosTill` as Long, `sText` as String)

Description

Replaces the text in the specified range.

Errors

- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

SelectionEnd

Property: `SelectionEnd` as Long

Description

Returns/sets the text selection end position.

Errors

- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

SelectionStart

Property: `SelectionStart` as Long

Description

Returns/sets the text selection start position.

Errors

- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

SelectText

Method: `SelectText`(`nPosFrom` as Long, `nPosTill` as Long)

Description

Selects the text in the specified range.

Errors

- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

SelText

Property: [SelText](#) as String

Description

Returns/sets the selected text.

Errors

- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

Text

Property: [Text](#) as String

Description

Returns/sets the document text.

Errors

- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

3.2.29 WSDL20DocumentationDlg

See also

Properties and Methods

Standard automation properties

[Application](#)

[Parent](#)

Interaction and visibility properties

[OptionsDialogAction](#)

[OutputFile](#)

[OutputFileDialogAction](#)

[ShowProgressBar](#)

[ShowResult](#)

Document generation options and methods

[OutputFormat](#)

[UseFixedDesign](#)

[SPSFile](#)

[EmbedDiagrams](#)

[DiagramFormat](#)

[MultipleOutputFiles](#)

[EmbedCSSInHTML](#)

[CreateDiagramsFolder](#)

[IncludeAll](#)

[IncludeBinding](#)

[IncludeImportedWSDLFiles](#)

[IncludeInterface](#)

[IncludeOverview](#)
[IncludeService](#)
[IncludeTypes](#)

[AllDetails](#)
[ShowBindingDiagram](#)
[ShowExtensibility](#)
[ShowEndpoint](#)
[ShowFault](#)
[ShowInterfaceDiagram](#)
[ShowOperation](#)
[ShowServiceDiagram](#)
[ShowSourceCode](#)
[ShowTypesDiagram](#)
[ShowUsedBy](#)

Description

This object combines all options for WSDL document generation as they are available through user interface dialog boxes in XMLSpy. The document generation options are initialized with the values used during the last generation of WSDL documentation. However, before using the object you have to set the [OutputFile](#) property to a valid file path. Use [OptionsDialogAction](#), [OutputFileDialogAction](#) and [ShowProgressBar](#) to specify the level of user interaction desired. You can use [HideAll](#) and [AllDetails](#) to set whole option groups at once or the individual properties to operate on a finer granularity.

AllDetails

See also

Method: [AllDetails](#) ([i_bDetailsOn](#) as Boolean)

Description

Use this method to turn all details options on or off.

Errors

4300 The object is no longer valid.

Application

See also

Property: [Application](#) as [Application](#) (read-only)

Description

Access the XMLSpy application object.

Errors

4300 The object is no longer valid.

4301 Invalid address for the return parameter was specified.

CreateDiagramsFolder

See also

Property: [CreateDiagramsFolder](#) as Boolean

Description

Set this property to `true`, to create a directory for the created images. Otherwise the diagrams will be created next to the documentation. This property is only available when the diagrams are not embedded. The property is initialized with the value used during the last call to [DocumentGenerateWSDL20Documentation](#). The default for the first run is false.

Errors

- 4300 The object is no longer valid.
- 4301 Invalid address for the return parameter was specified.

DiagramFormat

See also

Property: [DiagramFormat](#) as [SPYImageKind](#)

Description

This property specifies the generated diagram image type. This property is not available for HTML documentation. The property is initialized with the value used during the last call to [DocumentGenerateWSDL20Documentation](#). The default for the first run is PNG.

Errors

- 4300 The object is no longer valid.
- 4301 Invalid address for the return parameter was specified.

EmbedCSSInHTML

See also

Property: [EmbedCSSInHTML](#) as Boolean

Description

Set this property to `true`, to embed the CSS data in the generated HTML document. Otherwise a separate file will be created and linked. This property is only available for HTML documentation. The property is initialized with the value used during the last call to [DocumentGenerateWSDL20Documentation](#). The default for the first run is true.

Errors

- 4300 The object is no longer valid.
- 4301 Invalid address for the return parameter was specified.

EmbedDiagrams

See also

Property: [EmbedDiagrams](#) as Boolean

Description

Set this property to `true`, to embed the diagrams in the generated document. This property is not available for HTML documentation. The property is initialized with the value used during the last call to [DocumentGenerateWSDL20Documentation](#). The default for the first run is true.

Errors

- 4300 The object is no longer valid.
- 4301 Invalid address for the return parameter was specified.

IncludeAll**See also**

Method: [IncludeAll](#) (`i_bInclude` as `Boolean`)

Description

Use this method to mark or unmark all include options.

Errors

- 4300 The object is no longer valid.

IncludeBinding**See also**

Property: [IncludeBinding](#) as `Boolean`

Description

Set this property to `true`, to include bindings in the WSDL documentation. The property is initialized with the value used during the last call to [DocumentGenerateWSDL20Documentation](#). The default for the first run is true.

Errors

- 4300 The object is no longer valid.
- 4301 Invalid address for the return parameter was specified.

IncludeImportedWSDLFiles**See also**

Property: [IncludeImportedWSDLFiles](#) as `Boolean`

Description

Set this property to `true`, to include imported WSDL files in the WSDL documentation. The property is initialized with the value used during the last call to [DocumentGenerateWSDL20Documentation](#). The default for the first run is true.

Errors

- 4300 The object is no longer valid.
- 4301 Invalid address for the return parameter was specified.

IncludeInterface

See also

Property: [IncludeInterface](#) as Boolean

Description

Set this property to `true`, to include interfaces in the WSDL documentation. The property is initialized with the value used during the last call to [DocumentGenerateWSDL20Documentation](#). The default for the first run is true.

Errors

- 4300 The object is no longer valid.
- 4301 Invalid address for the return parameter was specified.

IncludeOverview

See also

Property: [IncludeOverview](#) as Boolean

Description

Set this property to `true`, to include an overview in the WSDL documentation. The property is initialized with the value used during the last call to [DocumentGenerateWSDL20Documentation](#). The default for the first run is true.

Errors

- 4300 The object is no longer valid.
- 4301 Invalid address for the return parameter was specified.

IncludeService

See also

Property: [IncludeService](#) as Boolean

Description

Set this property to `true`, to include services in the WSDL documentation. The property is initialized with the value used during the last call to [DocumentGenerateWSDL20Documentation](#). The default for the first run is true.

Errors

- 4300 The object is no longer valid.
- 4301 Invalid address for the return parameter was specified.

IncludeTypes

See also

Property: [IncludeTypes](#) as Boolean

Description

Set this property to `true`, to include types in the WSDL documentation. The property is initialized with the value used during the last call to [DocumentGenerateWSDL20Documentation](#). The default for the first run is true.

Errors

- 4300 The object is no longer valid.
- 4301 Invalid address for the return parameter was specified.

MultipleOutputFiles**See also**

Property: [MultipleOutputFiles](#) as Boolean

Description

Set this property to `true`, to split the documentation files. The property is initialized with the value used during the last call to [DocumentGenerateW_SDL20Documentation](#). The default for the first run is false.

Errors

- 4300 The object is no longer valid.
- 4301 Invalid value has been used to set the property.
Invalid address for the return parameter was specified.

OptionsDialogAction**See also**

Property: [OptionsDialogAction](#) as [SPYDAbgAction](#)

Description

To allow your script to fill in the default values and let the user see and react on the dialog, set this property to the value `spyDialogUserInput(2)`. If you want your script to define all the options in the schema documentation dialog without any user interaction necessary, use `spyDialogOK(0)`. Default is `spyDialogOK`.

Errors

- 4300 The object is no longer valid.
- 4301 Invalid value has been used to set the property.
Invalid address for the return parameter was specified.

OutputFile**See also**

Property: [OutputFile](#) as String

Description

Full path and name of the file that will contain the generated documentation. In case of HTML output, additional '.png' files will be generated based on this filename. The default value for this property is an empty string and needs to be replaced before using this object in a call to [DocumentGenerateW_SDL20Documentation](#).

Errors

- 4300 The object is no longer valid.
- 4301 Invalid address for the return parameter was specified.

OutputFileDialogAction

See also

Property: `OutputFileDialogAction` as [SPYDialogAction](#)

Description

To allow the user to select the output file with a file selection dialog, set this property to `spyDialogUserInput(2)`. If the value stored in [OutputFile](#) should be taken and no user interaction should occur, use `spyDialogOK(0)`. Default is `spyDialogOK`.

Errors

- 4300 The object is no longer valid.
- 4301 Invalid value has been used to set the property.
Invalid address for the return parameter was specified.

OutputFormat

See also

Property: `OutputFormat` as [SPYSchemaDocumentationFormat](#)

Description

Defines the kind of documentation that will be generated: HTML (value=0), MS-Word (value=1), or RTF (value=2). The property gets initialized with the value used during the last call to [DocumentGenerateWSDL20Documentation](#). The default for the first run is HTML.

Errors

- 4300 The object is no longer valid.
- 4301 Invalid value has been used to set the property.
Invalid address for the return parameter was specified.

Parent

See also

Property: `Parent` as [Dialogs](#) (read-only)

Description

Access the parent of the object.

Errors

- 4300 The object is no longer valid.
- 4301 Invalid address for the return parameter was specified.

ShowBindingDiagram

See also

Property: `ShowBindingDiagram` as `Boolean`

Description

Set this property to `true`, to show binding diagrams in the WSDL documentation. The property is initialized with the value used during the last call to [DocumentGenerateWSDL20Documentation](#). The default for the first run is `true`.

Errors

- 4300 The object is no longer valid.
- 4301 Invalid address for the return parameter was specified.

ShowEndpoint**See also**

Property: [ShowEndpoint](#) as Boolean

Description

Set this property to `true`, to show service endpoints in the WSDL documentation. The property is initialized with the value used during the last call to [DocumentGenerateWSDL20Documentation](#). The default for the first run is true.

Errors

- 4300 The object is no longer valid.
- 4301 Invalid address for the return parameter was specified.

ShowExtensibility**See also**

Property: [ShowExtensibility](#) as Boolean

Description

Set this property to `true`, to show service and binding extensibilities in the WSDL documentation. The property is initialized with the value used during the last call to [DocumentGenerateWSDL20Documentation](#). The default for the first run is true.

Errors

- 4300 The object is no longer valid.
- 4301 Invalid address for the return parameter was specified.

ShowFault**See also**

Property: [ShowFault](#) as Boolean

Description

Set this property to `true`, to show faults in the WSDL documentation. The property is initialized with the value used during the last call to [DocumentGenerateWSDL20Documentation](#). The default for the first run is true.

Errors

- 4300 The object is no longer valid.
- 4301 Invalid address for the return parameter was specified.

ShowInterfaceDiagram

See also

Property: [ShowInterfaceDiagram](#) as Boolean

Description

Set this property to `true`, to show interface diagrams in the WSDL documentation. The property is initialized with the value used during the last call to [DocumentGeneratorWSDL20Documentation](#). The default for the first run is true.

Errors

- 4300 The object is no longer valid.
- 4301 Invalid address for the return parameter was specified.

ShowOperation

See also

Property: [ShowOperation](#) as Boolean

Description

Set this property to `true`, to show interface and binding operations in the WSDL documentation. The property is initialized with the value used during the last call to [DocumentGeneratorWSDL20Documentation](#). The default for the first run is true.

Errors

- 4300 The object is no longer valid.
- 4301 Invalid address for the return parameter was specified.

ShowProgressBar

See also

Property: [ShowProgressBar](#) as Boolean

Description

Set this property to `true`, to make the window showing the document generation progress visible. Use `false`, to hide it. Default is `false`.

Errors

- 4300 The object is no longer valid.
- 4301 Invalid address for the return parameter was specified.

ShowResult

See also

Property: [ShowResult](#) as Boolean

Description

Set this property to `true`, to automatically open the resulting document when generation was successful. HTML documentation will be opened in XMLSpy. To show Word documentation,

MS-Word will be started. The property gets initialized with the value used during the last call to [DocumentGenerateWSDL20Documentation](#). The default for the first run is true.

Errors

- 4300 The object is no longer valid.
- 4301 Invalid address for the return parameter was specified.

ShowServiceDiagram**See also**

Property: [ShowServiceDiagram](#) as Boolean

Description

Set this property to `true`, to show service diagrams in the WSDL documentation. The property is initialized with the value used during the last call to [DocumentGenerateWSDL20Documentation](#). The default for the first run is true.

Errors

- 4300 The object is no longer valid.
- 4301 Invalid address for the return parameter was specified.

ShowSourceCode**See also**

Property: [ShowSourceCode](#) as Boolean

Description

Set this property to `true`, to show source code for the includes in the WSDL documentation. The property is initialized with the value used during the last call to [DocumentGenerateWSDL20Documentation](#). The default for the first run is true.

Errors

- 4300 The object is no longer valid.
- 4301 Invalid address for the return parameter was specified.

ShowTypesDiagram**See also**

Property: [ShowTypesDiagram](#) as Boolean

Description

Set this property to `true`, to show type diagrams in the WSDL documentation. The property is initialized with the value used during the last call to [DocumentGenerateWSDL20Documentation](#). The default for the first run is true.

Errors

- 4300 The object is no longer valid.
- 4301 Invalid address for the return parameter was specified.

ShowUsedBy**See also**

Property: [ShowUsedBy](#) as Boolean

Description

Set this property to `true`, to show the used-by relation for types, bindings and messages definitions in the WSDL documentation. The property is initialized with the value used during the last call to [DocumentGenerateWSDL20Documentation](#). The default for the first run is true.

Errors

- 4300 The object is no longer valid.
- 4301 Invalid address for the return parameter was specified.

SPSFile**See also**

Property: [SPSFile](#) as String

Description

Full path and name of the SPS file that will be used to generate the documentation.

Errors

- 4300 The object is no longer valid.
- 4301 Invalid address for the return parameter was specified.

UseFixedDesign**See also**

Property: [UseFixedDesign](#) as Boolean

Description

Specifies whether the documentation should be created with a fixed design or with a design specified by a SPS file (which requires StyleVision).

Errors

- 4300 The object is no longer valid.
- 4301 Invalid address for the return parameter was specified.

3.2.30 WSDLDocumentationDlg**See also****Properties and Methods**

Standard automation properties

[Application](#)
[Parent](#)

Interaction and visibility properties

[OptionsDialogAction](#)
[OutputFile](#)
[OutputFileDialogAction](#)
[ShowProgressBar](#)
[ShowResult](#)

Document generation options and methods

[OutputFormat](#)
[UseFixedDesign](#)
[SPSFile](#)
[EmbedDiagrams](#)
[DiagramFormat](#)
[MultipleOutputFiles](#)
[EmbedCSSInHTML](#)
[CreateDiagramsFolder](#)

[IncludeAll](#)
[IncludeBinding](#)
[IncludeImportedWSDLFiles](#)
[IncludeMessages](#)
[IncludeOverview](#)
[IncludePortType](#)
[IncludeService](#)
[IncludeTypes](#)

[AllDetails](#)
[ShowBindingDiagram](#)
[ShowExtensibility](#)
[ShowMessageParts](#)
[ShowPort](#)
[ShowPortTypeDiagram](#)
[ShowPortTypeOperations](#)
[ShowServiceDiagram](#)
[ShowSourceCode](#)
[ShowTypesDiagram](#)
[ShowUsedBy](#)

Description

This object combines all options for WSDL document generation as they are available through user interface dialog boxes in XMLSpy. The document generation options are initialized with the values used during the last generation of WSDL documentation. However, before using the object you have to set the [OutputFile](#) property to a valid file path. Use [OptionsDialogAction](#), [OutputFileDialogAction](#) and [ShowProgressBar](#) to specify the level of user interaction desired. You can use [IncludeAll](#) and [AllDetails](#) to set whole option groups at once or the individual properties to operate on a finer granularity.

AllDetails**See also**

Method: [AllDetails](#) ([i_bDetailsOn](#) as Boolean)

Description

Use this method to turn all details options on or off.

Errors

4300 The object is no longer valid.

Application

See also

Property: [Application](#) as [Application](#) (read-only)

Description

Access the XMLSpy application object.

Errors

3900 The object is no longer valid.

3901 Invalid address for the return parameter was specified.

CreateDiagramsFolder

See also

Property: [CreateDiagramsFolder](#) as `Boolean`

Description

Set this property to `true`, to create a directory for the created images. Otherwise the diagrams will be created next to the documentation. This property is only available when the diagrams are not embedded. The property is initialized with the value used during the last call to [DocumentGenerateWSDLDocumentation](#). The default for the first run is false.

Errors

3900 The object is no longer valid.

3901 Invalid address for the return parameter was specified.

DiagramFormat

See also

Property: [DiagramFormat](#) as [SPYImageKind](#)

Description

This property specifies the generated diagram image type. This property is not available for HTML documentation. The property is initialized with the value used during the last call to [DocumentGenerateWSDLDocumentation](#). The default for the first run is PNG.

Errors

3900 The object is no longer valid.

3901 Invalid address for the return parameter was specified.

EmbedCSSInHTML

See also

Property: [EmbedCSSInHTML](#) as Boolean

Description

Set this property to `true`, to embed the CSS data in the generated HTML document. Otherwise a separate file will be created and linked. This property is only available for HTML documentation. The property is initialized with the value used during the last call to [DocumentGenerateWSDLDocumentation](#). The default for the first run is true.

Errors

- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

EmbedDiagrams

See also

Property: [EmbedDiagrams](#) as Boolean

Description

Set this property to `true`, to embed the diagrams in the generated document. This property is not available for HTML documentation. The property is initialized with the value used during the last call to [DocumentGenerateWSDLDocumentation](#). The default for the first run is true.

Errors

- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

IncludeAll

See also

Method: [IncludeAll](#) ([i_bInclude](#) as Boolean)

Description

Use this method to mark or unmark all include options.

Errors

- 4300 The object is no longer valid.

IncludeBinding

See also

Property: [IncludeBinding](#) as Boolean

Description

Set this property to `true`, to include bindings in the WSDL documentation. The property is initialized with the value used during the last call to [DocumentGenerateWSDLDocumentation](#). The default for the first run is true.

Errors

- 3900 The object is no longer valid.

3901 Invalid address for the return parameter was specified.

IncludeImportedWSDLFiles

See also

Property: `IncludeImportedWSDLFiles` as `Boolean`

Description

Set this property to `true`, to include imported WSDL files in the WSDL documentation. The property is initialized with the value used during the last call to [DocumentGenerateWSDLDocumentation](#). The default for the first run is true.

Errors

3900 The object is no longer valid.
3901 Invalid address for the return parameter was specified.

IncludeMessages

See also

Property: `IncludeMessages` as `Boolean`

Description

Set this property to `true`, to include messages in the WSDL documentation. The property is initialized with the value used during the last call to [DocumentGenerateWSDLDocumentation](#). The default for the first run is true.

Errors

3900 The object is no longer valid.
3901 Invalid address for the return parameter was specified.

IncludeOverview

See also

Property: `IncludeOverview` as `Boolean`

Description

Set this property to `true`, to include an overview in the WSDL documentation. The property is initialized with the value used during the last call to [DocumentGenerateWSDLDocumentation](#). The default for the first run is true.

Errors

3900 The object is no longer valid.
3901 Invalid address for the return parameter was specified.

IncludePortType

See also

Property: `IncludePortType` as `Boolean`

Description

Set this property to `true`, to include port types in the WSDL documentation. The property is

initialized with the value used during the last call to [DocumentGenerateWSDLDocumentation](#). The default for the first run is true.

Errors

- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

IncludeService**See also**

Property: [IncludeService](#) as Boolean

Description

Set this property to `true`, to include services in the WSDL documentation. The property is initialized with the value used during the last call to [DocumentGenerateWSDLDocumentation](#). The default for the first run is true.

Errors

- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

IncludeTypes**See also**

Property: [IncludeTypes](#) as Boolean

Description

Set this property to `true`, to include types in the WSDL documentation. The property is initialized with the value used during the last call to [DocumentGenerateWSDLDocumentation](#). The default for the first run is true.

Errors

- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

MultipleOutputFiles**See also**

Property: [MultipleOutputFiles](#) as Boolean

Description

Set this property to `true`, to split the documentation files. The property is initialized with the value used during the last call to [DocumentGenerateWSDLDocumentation](#). The default for the first run is false.

Errors

- 3900 The object is no longer valid.
- 3901 Invalid value has been used to set the property.
Invalid address for the return parameter was specified.

OptionsDialogAction

See also

Property: [OptionsDialogAction](#) as [SPYDialogAction](#)

Description

To allow your script to fill in the default values and let the user see and react on the dialog, set this property to the value *spyDialogUserInput(2)*. If you want your script to define all the options in the schema documentation dialog without any user interaction necessary, use *spyDialogOK(0)*. Default is *spyDialogOK*.

Errors

- 3900 The object is no longer valid.
- 3901 Invalid value has been used to set the property.
Invalid address for the return parameter was specified.

OutputFile

See also

Property: [OutputFile](#) as [String](#)

Description

Full path and name of the file that will contain the generated documentation. In case of HTML output, additional '.png' files will be generated based on this filename. The default value for this property is an empty string and needs to be replaced before using this object in a call to [DocumentGenerateWSDLDocumentation](#).

Errors

- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

OutputFileDialogAction

See also

Property: [OutputFileDialogAction](#) as [SPYDialogAction](#)

Description

To allow the user to select the output file with a file selection dialog, set this property to *spyDialogUserInput(2)*. If the value stored in [OutputFile](#) should be taken and no user interaction should occur, use *spyDialogOK(0)*. Default is *spyDialogOK*.

Errors

- 3900 The object is no longer valid.
- 3901 Invalid value has been used to set the property.
Invalid address for the return parameter was specified.

OutputFormat

See also

Property: [OutputFormat](#) as [SPYSchemaDocumentationFormat](#)

Description

Defines the kind of documentation that will be generated: HTML (value=0), MS-Word (value=1), or RTF (value=2). The property gets initialized with the value used during the last call to [DocumentGenerateWSDLDocumentation](#). The default for the first run is HTML.

Errors

- 3900 The object is no longer valid.
- 3901 Invalid value has been used to set the property.
Invalid address for the return parameter was specified.

Parent**See also**

Property: [Parent](#) as [Dialogs](#) (read-only)

Description

Access the parent of the object.

Errors

- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

ShowBindingDiagram**See also**

Property: [ShowBindingDiagram](#) as [Boolean](#)

Description

Set this property to `true`, to show binding diagrams in the WSDL documentation. The property is initialized with the value used during the last call to [DocumentGenerateWSDLDocumentation](#). The default for the first run is true.

Errors

- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

ShowExtensibility**See also**

Property: [ShowExtensibility](#) as [Boolean](#)

Description

Set this property to `true`, to show service and binding extensibilities in the WSDL documentation. The property is initialized with the value used during the last call to [DocumentGenerateWSDLDocumentation](#). The default for the first run is true.

Errors

- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

ShowMessageParts

See also

Property: [ShowMessageParts](#) as Boolean

Description

Set this property to `true`, to show message parts of messages in the WSDL documentation. The property is initialized with the value used during the last call to [DocumentGenerateWSDLDocumentation](#). The default for the first run is true.

Errors

- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

ShowPort

See also

Property: [ShowPort](#) as Boolean

Description

Set this property to `true`, to show service ports in the WSDL documentation. The property is initialized with the value used during the last call to [DocumentGenerateWSDLDocumentation](#). The default for the first run is true.

Errors

- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

ShowPortTypeDiagram

See also

Property: [ShowPortTypeDiagram](#) as Boolean

Description

Set this property to `true`, to show port type diagrams in the WSDL documentation. The property is initialized with the value used during the last call to [DocumentGenerateWSDLDocumentation](#). The default for the first run is true.

Errors

- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

ShowPortTypeOperations

See also

Property: [ShowPortTypeOperations](#) as Boolean

Description

Set this property to `true`, to show port type operations in the WSDL documentation. The property is initialized with the value used during the last call to [DocumentGenerateWSDLDocumentation](#). The default for the first run is true.

Errors

- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

ShowProgressBar**See also**

Property: [ShowProgressBar](#) as `Boolean`

Description

Set this property to `true`, to make the window showing the document generation progress visible. Use `false`, to hide it. Default is `false`.

Errors

- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

ShowResult**See also**

Property: [ShowResult](#) as `Boolean`

Description

Set this property to `true`, to automatically open the resulting document when generation was successful. HTML documentation will be opened in XMLSpy. To show Word documentation, MS-Word will be started. The property gets initialized with the value used during the last call to [DocumentGenerateWSDLDocumentation](#). The default for the first run is true.

Errors

- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

ShowServiceDiagram**See also**

Property: [ShowServiceDiagram](#) as `Boolean`

Description

Set this property to `true`, to show service diagrams in the WSDL documentation. The property is initialized with the value used during the last call to [DocumentGenerateWSDLDocumentation](#). The default for the first run is true.

Errors

- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

ShowSourceCode

See also

Property: [ShowSourceCode](#) as Boolean

Description

Set this property to `true`, to show source code for the includes in the WSDL documentation. The property is initialized with the value used during the last call to [DocumentGenerateWSDLDocumentation](#). The default for the first run is true.

Errors

- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

ShowTypesDiagram

See also

Property: [ShowTypesDiagram](#) as Boolean

Description

Set this property to `true`, to show type diagrams in the WSDL documentation. The property is initialized with the value used during the last call to [DocumentGenerateWSDLDocumentation](#). The default for the first run is true.

Errors

- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

ShowUsedBy

See also

Property: [ShowUsedBy](#) as Boolean

Description

Set this property to `true`, to show the used-by relation for types, bindings and messages definitions in the WSDL documentation. The property is initialized with the value used during the last call to [DocumentGenerateWSDLDocumentation](#). The default for the first run is true.

Errors

- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

UseFixedDesign

See also

Property: [UseFixedDesign](#) as Boolean

Description

Specifies whether the documentation should be created with a fixed design or with a design specified by a SPS file (which requires StyleVision).

Errors

- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

SPSFile**See also**

Property: [SPSFile](#) as String

Description

Full path and name of the SPS file that will be used to generate the documentation.

Errors

- 3900 The object is no longer valid.
- 3901 Invalid address for the return parameter was specified.

3.2.31 XBRLDocumentationDlg

See also**Properties and Methods**

Standard automation properties

[Application](#)

[Parent](#)

Interaction and visibility properties

[OptionsDialogAction](#)

[OutputFile](#)

[OutputFileDialogAction](#)

[ShowProgressBar](#)

[ShowResult](#)

Document generation options and methods

[OutputFormat](#)

[UseFixedDesign](#)

[SPSFile](#)

[EmbedDiagrams](#)

[DiagramFormat](#)

[EmbedCSSInHTML](#)

[CreateDiagramsFolder](#)

[IncludeAll](#)

[IncludeOverview](#)

[IncludeNamespacePrefixes](#)

[IncludeGlobalElements](#)

[IncludeDefinitionLinkroles](#)

[IncludePresentationLinkroles](#)

[IncludeCalculationLinkroles](#)

[AllDetails](#)

[ShowDiagram](#)
[ShowSubstitutiongroup](#)
[ShowItemtype](#)
[ShowBalance](#)
[ShowPeriod](#)
[ShowAbstract](#)
[ShowNillable](#)
[ShowLabels](#)
[ShowReferences](#)
[ShowLinkbaseReferences](#)

[ShortQualifiedNames](#)
[ShowImportedElements](#)

Description

This object combines all options for XBRL document generation as they are available through user interface dialog boxes in XMLSpy. The document generation options are initialized with the values used during the last generation of XBRL documentation. However, before using the object you have to set the [OutputFile](#) property to a valid file path. Use [OptionsDialogAction](#), [OutputFileDialogAction](#) and [ShowProgressBar](#) to specify the level of user interaction desired. You can use [AllOptions](#) and [AllDetails](#) to set whole option groups at once or the individual properties to operate on a finer granularity.

AllDetails

See also

Method: [AllDetails](#) ([i_bDetailsOn](#) as Boolean)

Description

Use this method to turn all details options on or off.

Errors

4400 The object is no longer valid.

Application

See also

Property: [Application](#) as [Application](#) (read-only)

Description

Access the XMLSpy application object.

Errors

4400 The object is no longer valid.

4401 Invalid address for the return parameter was specified.

CreateDiagramsFolder

See also

Property: [CreateDiagramsFolder](#) as Boolean

Description

Set this property to `true`, to create a directory for the created images. Otherwise the diagrams will be created next to the documentation. This property is only available when the diagrams are not embedded. The property is initialized with the value used during the last call to [DocumentGenerateXBRLDocumentation](#). The default for the first run is false.

Errors

- 4400 The object is no longer valid.
- 4401 Invalid address for the return parameter was specified.

DiagramFormat

See also

Property: [DiagramFormat](#) as [SPYImageKind](#)

Description

This property specifies the generated diagram image type. This property is not available for HTML documentation. The property is initialized with the value used during the last call to [DocumentGenerateXBRLDocumentation](#). The default for the first run is PNG.

Errors

- 4400 The object is no longer valid.
- 4401 Invalid address for the return parameter was specified.

EmbedCSSInHTML

See also

Property: [EmbedCSSInHTML](#) as Boolean

Description

Set this property to `true`, to embed the CSS data in the generated HTML document. Otherwise a separate file will be created and linked. This property is only available for HTML documentation. The property is initialized with the value used during the last call to [DocumentGenerateXBRLDocumentation](#). The default for the first run is true.

Errors

- 4400 The object is no longer valid.
- 4401 Invalid address for the return parameter was specified.

EmbedDiagrams

See also

Property: [EmbedDiagrams](#) as Boolean

Description

Set this property to `true`, to embed the diagrams in the generated document. This property is not available for HTML documentation. The property is initialized with the value used during the last call to [DocumentGenerateXBRLDocumentation](#). The default for the first run is true.

Errors

- 4400 The object is no longer valid.
- 4401 Invalid address for the return parameter was specified.

IncludeAll**See also**

Method: [IncludeAll](#) (`i_bInclude` as `Boolean`)

Description

Use this method to mark or unmark all include options.

Errors

- 4400 The object is no longer valid.

IncludeCalculationLinkroles**See also**

Property: [IncludeCalculationLinkroles](#) as `Boolean`

Description

Set this property to `true`, to include calculation linkroles in the XBRL documentation. The property is initialized with the value used during the last call to [DocumentGenerateXBRLDocumentation](#). The default for the first run is true.

Errors

- 4400 The object is no longer valid.
- 4401 Invalid address for the return parameter was specified.

IncludeDefinitionLinkroles**See also**

Property: [IncludeDefinitionLinkroles](#) as `Boolean`

Description

Set this property to `true`, to include definition linkroles in the XBRL documentation. The property is initialized with the value used during the last call to [DocumentGenerateXBRLDocumentation](#). The default for the first run is true.

Errors

- 4400 The object is no longer valid.
- 4401 Invalid address for the return parameter was specified.

IncludeGlobalElements

See also

Property: [IncludeGlobalElements](#) as Boolean

Description

Set this property to `true`, to include global elements in the XBRL documentation. The property is initialized with the value used during the last call to [DocumentGenerateXBRLDocumentation](#). The default for the first run is true.

Errors

- 4400 The object is no longer valid.
- 4401 Invalid address for the return parameter was specified.

IncludeNamespacePrefixes

See also

Property: [IncludeNamespacePrefixes](#) as Boolean

Description

Set this property to `true`, to include namespace prefixes in the XBRL documentation. The property is initialized with the value used during the last call to [DocumentGenerateXBRLDocumentation](#). The default for the first run is true.

Errors

- 4400 The object is no longer valid.
- 4401 Invalid address for the return parameter was specified.

IncludeOverview

See also

Property: [IncludeOverview](#) as Boolean

Description

Set this property to `true`, to include an overview in the XBRL documentation. The property is initialized with the value used during the last call to [DocumentGenerateXBRLDocumentation](#). The default for the first run is true.

Errors

- 4400 The object is no longer valid.
- 4401 Invalid address for the return parameter was specified.

IncludePresentationLinkroles

See also

Property: [IncludePresentationLinkroles](#) as Boolean

Description

Set this property to `true`, to include presentation linkroles in the XBRL documentation. The property is initialized with the value used during the last call to [DocumentGenerateXBRLDocumentation](#). The default for the first run is true.

Errors

- 4400 The object is no longer valid.
- 4401 Invalid address for the return parameter was specified.

OptionsDialogAction**See also**

Property: [OptionsDialogAction](#) as [SPYDialogAction](#)

Description

To allow your script to fill in the default values and let the user see and react on the dialog, set this property to the value *spyDialogUserInput(2)*. If you want your script to define all the options in the schema documentation dialog without any user interaction necessary, use *spyDialogOK(0)*. Default is *spyDialogOK*.

Errors

- 4400 The object is no longer valid.
- 4401 Invalid value has been used to set the property.
Invalid address for the return parameter was specified.

OutputFile**See also**

Property: [OutputFile](#) as [String](#)

Description

Full path and name of the file that will contain the generated documentation. In case of HTML output, additional '.png' files will be generated based on this filename. The default value for this property is an empty string and needs to be replaced before using this object in a call to [DocumentGenerateXBRLDocumentation](#).

Errors

- 4400 The object is no longer valid.
- 4401 Invalid address for the return parameter was specified.

OutputFileDialogAction**See also**

Property: [OutputFileDialogAction](#) as [SPYDialogAction](#)

Description

To allow the user to select the output file with a file selection dialog, set this property to *spyDialogUserInput(2)*. If the value stored in [OutputFile](#) should be taken and no user interaction should occur, use *spyDialogOK(0)*. Default is *spyDialogOK*.

Errors

- 4400 The object is no longer valid.
- 4401 Invalid value has been used to set the property.
Invalid address for the return parameter was specified.

OutputFormat

See also

Property: `OutputFormat` as [SPYSchemaDocumentationFormat](#)

Description

Defines the kind of documentation that will be generated: HTML (value=0), MS-Word (value=1), or RTF (value=2). The property gets initialized with the value used during the last call to [DocumentGenerateXBRLDocumentation](#). The default for the first run is HTML.

Errors

- 4400 The object is no longer valid.
- 4401 Invalid value has been used to set the property.
Invalid address for the return parameter was specified.

Parent

See also

Property: `Parent` as [Dialogs](#) (read-only)

Description

Access the parent of the object.

Errors

- 4400 The object is no longer valid.
- 4401 Invalid address for the return parameter was specified.

ShortQualifiedNames

See also

Property: `ShortQualifiedNames` as `Boolean`

Description

Set this property to `true`, to use short qualified names in the XBRL documentation. The property is initialized with the value used during the last call to [DocumentGenerateXBRLDocumentation](#). The default for the first run is `true`.

Errors

- 4400 The object is no longer valid.
- 4401 Invalid address for the return parameter was specified.

ShowAbstracts

See also

Property: `ShowAbstracts` as `Boolean`

Description

Set this property to `true`, to show abstracts in the XBRL documentation. The property is initialized with the value used during the last call to [DocumentGenerateXBRLDocumentation](#). The default for the first run is `true`.

Errors

- 4400 The object is no longer valid.
- 4401 Invalid address for the return parameter was specified.

ShowBalance**See also**

Property: [ShowBalance](#) as Boolean

Description

Set this property to `true`, to show balances in the XBRL documentation. The property is initialized with the value used during the last call to [DocumentGenerateXBRLDocumentation](#). The default for the first run is true.

Errors

- 4400 The object is no longer valid.
- 4401 Invalid address for the return parameter was specified.

ShowDiagram**See also**

Property: [ShowDiagram](#) as Boolean

Description

Set this property to `true`, to show diagrams in the XBRL documentation. The property is initialized with the value used during the last call to [DocumentGenerateXBRLDocumentation](#). The default for the first run is true.

Errors

- 4400 The object is no longer valid.
- 4401 Invalid address for the return parameter was specified.

ShowImportedElements**See also**

Property: [ShowImportedElements](#) as Boolean

Description

Set this property to `true`, to show imported elements in the XBRL documentation. The property is initialized with the value used during the last call to [DocumentGenerateXBRLDocumentation](#). The default for the first run is true.

Errors

- 4400 The object is no longer valid.
- 4401 Invalid address for the return parameter was specified.

ShowItemtype

See also

Property: [ShowItemtype](#) as Boolean

Description

Set this property to `true`, to show item types in the XBRL documentation. The property is initialized with the value used during the last call to [DocumentGenerateXBRLDocumentation](#). The default for the first run is true.

Errors

- 4400 The object is no longer valid.
- 4401 Invalid address for the return parameter was specified.

ShowLabels

See also

Property: [ShowLabels](#) as Boolean

Description

Set this property to `true`, to show labels in the XBRL documentation. The property is initialized with the value used during the last call to [DocumentGenerateXBRLDocumentation](#). The default for the first run is true.

Errors

- 4400 The object is no longer valid.
- 4401 Invalid address for the return parameter was specified.

ShowLinkbaseReferences

See also

Property: [ShowLinkbaseReferences](#) as Boolean

Description

Set this property to `true`, to show linkbase references in the XBRL documentation. The property is initialized with the value used during the last call to [DocumentGenerateXBRLDocumentation](#). The default for the first run is true.

Errors

- 4400 The object is no longer valid.
- 4401 Invalid address for the return parameter was specified.

ShowNillable

See also

Property: [ShowNillable](#) as Boolean

Description

Set this property to `true`, to show nillable properties in the XBRL documentation. The property is initialized with the value used during the last call to [DocumentGenerateXBRLDocumentation](#). The default for the first run is true.

Errors

- 4400 The object is no longer valid.
- 4401 Invalid address for the return parameter was specified.

ShowPeriod**See also**

Property: [ShowPeriod](#) as `Boolean`

Description

Set this property to `true`, to show periods in the XBRL documentation. The property is initialized with the value used during the last call to [DocumentGenerateXBRLDocumentation](#). The default for the first run is true.

Errors

- 4400 The object is no longer valid.
- 4401 Invalid address for the return parameter was specified.

ShowProgressBar**See also**

Property: [ShowProgressBar](#) as `Boolean`

Description

Set this property to `true`, to make the window showing the document generation progress visible. Use `false`, to hide it. Default is `false`.

Errors

- 4400 The object is no longer valid.
- 4401 Invalid address for the return parameter was specified.

ShowReferences**See also**

Property: [ShowReferences](#) as `Boolean`

Description

Set this property to `true`, to show references in the XBRL documentation. The property is initialized with the value used during the last call to [DocumentGenerateXBRLDocumentation](#). The default for the first run is true.

Errors

- 4400 The object is no longer valid.
- 4401 Invalid address for the return parameter was specified.

ShowResult

See also

Property: [ShowResult](#) as Boolean

Description

Set this property to `true`, to automatically open the resulting document when generation was successful. HTML documentation will be opened in XMLSpy. To show Word documentation, MS-Word will be started. The property gets initialized with the value used during the last call to [DocumentGenerateXBRLDocumentation](#). The default for the first run is true.

Errors

- 4400 The object is no longer valid.
- 4401 Invalid address for the return parameter was specified.

ShowSubstitutiongroup

See also

Property: [ShowSubstitutiongroup](#) as Boolean

Description

Set this property to `true`, to show substitution groups in the XBRL documentation. The property is initialized with the value used during the last call to [DocumentGenerateXBRLDocumentation](#). The default for the first run is true.

Errors

- 4400 The object is no longer valid.
- 4401 Invalid address for the return parameter was specified.

SPSFile

See also

Property: [SPSFile](#) as String

Description

Full path and name of the SPS file that will be used to generate the documentation.

Errors

- 4400 The object is no longer valid.
- 4401 Invalid address for the return parameter was specified.

UseFixedDesign

See also

Property: [UseFixedDesign](#) as Boolean

Description

Specifies whether the documentation should be created with a fixed design or with a design

specified by a SPS file (which requires StyleVision).

Errors

- 4400 The object is no longer valid.
- 4401 Invalid address for the return parameter was specified.

3.2.32 XMLData**See also****Properties**[Kind](#)[Name](#)[TextValue](#)[HasChildren](#)[MayHaveChildren](#)[Parent](#)**Methods**[GetFirstChild](#)[GetNextChild](#)[GetCurrentChild](#)[InsertChild](#)[InsertChildAfter](#)[InsertChildBefore](#)[AppendChild](#)[EraseAllChildren](#)[EraseChild](#)[EraseCurrentChild](#)[IsSameNode](#)[CountChildren](#)[CountChildrenKind](#)[GetChild](#)[GetChildAttribute](#)[GetChildElement](#)[GetChildKind](#)[GetNamespacePrefixForURI](#)[HasChildrenKind](#)[SetTextValueXMLEncoded](#)**Description**

The XMLData interface provides direct XML-level access to a document. You can read and directly modify the XML representation of the document. However, please, note the following restrictions:

- The XMLData representation is only valid when the document is shown in grid view or authentic view.
- When in authentic view, additional XMLData elements are automatically inserted as parents of each visible document element. Typically this is an XMLData of kind

`spyXMLDataElement` with the [Name](#) property set to 'Text'.

- When you use the XMLData interface while in a different view mode you will not receive errors, but changes are not reflected to the view and might get lost during the next view switch.

Note also:

- Setting a new text value for an XML element is possible if the element does not have non-text children. A text value can be set even if the element has attributes.
- When setting a new text value for an XML element which has more than one text child, the latter will be deleted and replaced by one new text child.
- When reading the text value of an XML element which has more than one text child, only the value of the first text child will be returned.

Objects of this class represent the different atomic parts of an XML document. See the enumeration type [SPYXMLDataKind](#) for the available part types. Each part knows its children, thus forming a XMLData tree with [DocumentRootElement](#) at its top. To get the top element of the document content - ignoring the XML header - use [DocumentDataRoot](#). For examples on how to traverse the XMLData tree see [Using XMLData to modify document structure](#) and [GetNextChild](#).

AppendChild

See also

Declaration: `AppendChild (pNewData as XMLData)`

Description

`AppendChild` appends `pNewData` as last child to the `XMLData` object. See also [Using XMLData](#).

Errors

- 1500 The XMLData object is no longer valid.
- 1505 Invalid XMLData kind was specified.
- 1506 Invalid address for the return parameter was specified.
- 1507 Element cannot have Children
- 1512 Cyclic insertion - new data element is already part of document
- 1514 Invalid XMLData kind was specified for this position.
- 1900 Document must not be modified

Example

```
Dim objCurrentParent As XMLData
Dim objNewChild As XMLData

Set objNewChild = objSpy.ActiveDocument.CreateChild(spyXMLDataElement)
Set objCurrentParent = objSpy.ActiveDocument.RootElement

objCurrentParent.AppendChild objNewChild

Set objNewChild = Nothing
```

CountChildren

See also

Declaration: `CountChildren` as long

Description

`CountChildren` gets the number of children.

Available with TypeLibrary version 1.5

Errors

1500 The XMLData object is no longer valid.

CountChildrenKind

See also

Declaration: `CountChildrenKind` (*nKind* as [SPYXMLDataKind](#)) as long

Description

`CountChildrenKind` gets the number of children of the specific kind.

Available with TypeLibrary version 1.5

Errors

1500 The XMLData object is no longer valid.

EraseAllChildren

See also

Declaration: `EraseAllChildren`

Description

`EraseAllChildren` deletes all associated children of the XMLData object.

Errors

1500 The XMLData object is no longer valid.

1900 Document must not be modified

Example

The sample erases all elements of the active document.

```
Dim objCurrentParent As XMLData

Set objCurrentParent = objSpy.ActiveDocument.RootElement
objCurrentParent.EraseAllChildren
```

EraseChild

Method: `EraseChild` (Child as [XMLData](#))

Description

Deletes the given child node.

Errors

1500 Invalid object.

1506 Invalid input xml

1510 Invalid parameter.

EraseCurrentChild

See also

Declaration: [EraseCurrentChild](#)

Description

[EraseCurrentChild](#) deletes the current [XMLData](#) child object. Before you call [EraseCurrentChild](#) you must initialize an internal iterator with [XMLData.GetFirstChild](#). After deleting the current child, [EraseCurrentChild](#) increments the internal iterator of the [XMLData](#) element. No error is returned when the last child gets erased and the iterator is moved past the end of the child list. The next call to [EraseCurrentChild](#) however, will return error 1503.

Errors

- 1500 The [XMLData](#) object is no longer valid.
- 1503 No iterator is initialized for this [XMLData](#) object, or the iterator points past the last child.
- 1900 Document must not be modified

Examples

```
// -----
// XMLSpy scripting environment - JScript
// erase all children of XMLData
// -----
// let's get an XMLData element, we assume that the
// cursor selects the parent of a list in grid view
var objList = Application.ActiveDocument.GridView.CurrentFocus;

// the following line would be shorter, of course
//objList.EraseAllChildren();

// but we want to demonstrate the usage of EraseCurrentChild
if ((objList != null) && (objList.HasChildren))
{
    try
    {
        objEle = objList.GetFirstChild(-1);
        while (objEle != null)
        {
            objList.EraseCurrentChild();
            // no need to call GetNextChild
        }
    }
    catch (err)
    {
        // 1503 - we reached end of child list
        if ((err.number & 0xffff) != 1503) throw (err);
    }
}
```

GetChild

See also

Declaration: [GetChild](#) (*position* as long) as [XMLData](#)

Return Value

Returns an XML element as [XMLData](#) object.

Description

[GetChild](#) returns a reference to the child at the given index (zero-based).

Available with TypeLibrary version 1.5

Errors

- 1500 The XMLData object is no longer valid.
- 1510 Invalid address for the return parameter was specified.

GetChildAttribute

Method: [GetChildAttribute](#) (strName as string) child as XMLData object (NULL on error)

Description

Retrieves the attribute having the given name.

Errors

- 1500 Invalid object.
- 1510 Invalid parameter.

GetChildElement

Method: [GetChildElement](#) (strName as string, nIndex as long) child as XMLData object (NULL on error)

Description

Retrieves the Nth child element with the given name.

Errors

- 1500 Invalid object.
- 1510 Invalid parameter.

GetChildKind**See also**

Declaration: [GetChildKind](#) (*position* as long, *nKind* as [SPYXMLDataKind](#)) as [XMLData](#)

Return Value

Returns an XML element as XMLData object.

Description

[GetChildKind](#) returns a reference to a child of this kind at the given index (zero-based). The position parameter is relative to the number of children of the specified kind and not to all children of the object.

Available with TypeLibrary version 1.5

Errors

- 1500 The XMLData object is no longer valid.
- 1510 Invalid address for the return parameter was specified.

GetCurrentChild

See also

Declaration: [GetCurrentChild](#) as [XMLData](#)

Return Value

Returns an XML element as `XMLData` object.

Description

[GetCurrentChild](#) gets the current child. Before you call [GetCurrentChild](#) you must initialize an internal iterator with [XMLData.GetFirstChild](#).

Errors

- 1500 The XMLData object is no longer valid.
- 1503 No iterator is initialized for this XMLData object.
- 1510 Invalid address for the return parameter was specified.

GetFirstChild

See also

Declaration: [GetFirstChild](#) (*nKind* as [SPYXMLDataKind](#)) as [XMLData](#)

Return Value

Returns an XML element as `XMLData` object.

Description

[GetFirstChild](#) initializes a new iterator and returns the first child. Set `nKind = -1` to get an iterator for all kinds of children.

REMARK: The iterator is stored inside the XMLData object and gets destroyed when the XMLData object gets destroyed. Be sure to keep a reference to this object as long as you want to use [GetCurrentChild](#), [GetNextChild](#) or [EraseCurrentChild](#).

Errors

- 1500 The XMLData object is no longer valid.
- 1501 Invalid XMLData kind was specified.
- 1504 Element has no children of specified kind.
- 1510 Invalid address for the return parameter was specified.

Example

See the example at [XMLData.GetNextChild](#).

GetNamespacePrefixForURI

Method: [GetNamespacePrefixForURI](#) (strURI as string) strNS as string

Description

Returns the namespace prefix of the supplied URI.

Errors

- 1500 Invalid object.
- 1510 Invalid parameter.

GetNextChild

See also

Declaration: `GetNextChild` as [XMLData](#)

Return Value

Returns an XML element as `XMLData` object.

Description

`GetNextChild` steps to the next child of this element. Before you call `GetNextChild` you must initialize an internal iterator with [XMLData.GetFirstChild](#).

Check for the last child of the element as shown in the sample below.

Errors

- 1500 The XMLData object is no longer valid.
- 1503 No iterator is initialized for this XMLData object.
- 1510 Invalid address for the return parameter was specified.

Examples

```
' -----
' VBA code snippet - iterate XMLData children
' -----
On Error Resume Next
Set objParent = objSpy.ActiveDocument.RootElement

'get elements of all kinds
Set objCurrentChild = objParent.GetFirstChild(-1)

Do
    'do something useful with the child

    'step to next child
    Set objCurrentChild = objParent.GetNextChild
Loop Until (Err.Number - vbObjectError = 1503)

// -----
// XMLSpy scripting environment - JScript
// iterate through children of XMLData
// -----
try
{
    var objXMLData = ... // initialize somehow
    var objChild = objXMLData.GetFirstChild(-1);

    while ( true)
    {
        // do something usefull with objChild

        objChild = objXMLData.GetNextChild();
    }
}
catch (err)
{
    if ((err.number & 0xffff) == 1504)
        ; // element has no children
}
```

```
    else if ((err.number & 0xffff) == 1503)
        ; // last child reached
    else
        throw (err);
}
```

GetTextValueXMLDecoded

Method: `GetTextValueXMLDecoded()` as string

Description

Gets the decoded text value of the XML.

Errors

- 1500 Invalid object.
- 1510 Invalid parameter.

HasChildren

See also

Declaration: `HasChildren` as Boolean

Description

The property is true if the object is the parent of other `XMLData` objects. This property is read-only.

Errors

- 1500 The `XMLData` object is no longer valid.
- 1510 Invalid address for the return parameter was specified.

HasChildrenKind

See also

Declaration: `HasChildrenKind` (*nKind* as [SPYXMLDataKind](#)) as Boolean

Description

The method returns true if the object is the parent of other `XMLData` objects of the specific kind.

Available with TypeLibrary version 1.5

Errors

- 1500 The `XMLData` object is no longer valid.
- 1510 Invalid address for the return parameter was specified.

InsertChild

See also

Declaration: `InsertChild` (*pNewData* as [XMLData](#))

Description

~~InsertChild~~ inserts the new child before the current child (see also [XMLData.GetFirstChild](#), [XMLData.GetNextChild](#) to set the current child).

Errors

- 1500 The XMLData object is no longer valid.
- 1503 No iterator is initialized for this XMLData object.
- 1505 Invalid XMLData kind was specified.
- 1506 Invalid address for the return parameter was specified.
- 1507 Element cannot have Children
- 1512 Cyclic insertion - new data element is already part of document
- 1514 Invalid XMLData kind was specified for this position.
- 1900 Document must not be modified

Examples

See [Using XMLData to modify document structure](#) .

InsertChildAfter

Method: [InsertChildBefore](#) (Node as XMLData, NewData as XMLData)

Description

Inserts a new XML node (supplied with the second parameter) after the specified node (first parameter).

Errors

- 1500 Invalid object.
- 1506 Invalid input xml
- 1507 No children allowed
- 1510 Invalid parameter.
- 1512 Child is already added
- 1514 Invalid kind at position

InsertChildBefore

Method: [InsertChildBefore](#) (Node as XMLData, NewData as XMLData)

Description

Inserts a new XML node (supplied with the second parameter) before the specified node (first parameter).

Errors

- 1500 Invalid object.
- 1506 Invalid input xml
- 1507 No children allowed
- 1510 Invalid parameter.
- 1512 Child is already added
- 1514 Invalid kind at position

IsSameNode**See also**

Declaration: `IsSameNode (pNodeToCompare as XMLData) as Boolean`

Description

Returns true if `pNodeToCompare` references the same node as the object itself.

Errors

- 1500 The XMLData object is no longer valid.
- 1506 Invalid address for the return parameter was specified.

Kind**See also**

Declaration: `Kind as SPYXMLDataKind`

Description

Kind of this `XMLData` object. This property is read-only.

Errors

- 1500 The XMLData object is no longer valid.
- 1510 Invalid address for the return parameter was specified.

MayHaveChildren**See also**

Declaration: `MayHaveChildren as Boolean`

Description

Indicates whether it is allowed to add children to this `XMLData` object. This property is read-only.

Errors

- 1500 The XMLData object is no longer valid.
- 1510 Invalid address for the return parameter was specified.

Name**See also**

Declaration: `Name as String`

Description

Used to modify and to get the name of the `XMLData` object.

Errors

- 1500 The XMLData object is no longer valid.
- 1510 Invalid address for the return parameter was specified.

Parent**See also**

Declaration: `Parent` as [XMLData](#)

Return value

Parent as `XMLData` object. Nothing (or NULL) if there is no parent element.

Description

Parent of this element. This property is read-only.

Errors

- 1500 The XMLData object is no longer valid.
- 1510 Invalid address for the return parameter was specified.

SetTextValueXMLEncoded

Method: `SetTextValueXMLEncoded` (`strVal` as [String](#))

Description

Sets the encoded text value of the XML.

Errors

- 1500 Invalid object.
- 1513 Modification not allowed.

TextValue**See also**

Declaration: `TextValue` as `String`

Description

Used to modify and to get the text value of this `XMLData` object.

Errors

- 1500 The XMLData object is no longer valid.
- 1510 Invalid address for the return parameter was specified.

Examples

See [Using XMLData to modify document structure](#) .

3.3 Interfaces (obsolete)

Interfaces contained in this book are obsolete. It is recommended to migrate your applications to the new interfaces. See the different properties and methods in this book for migration hints.

3.3.1 AuthenticEvent (obsolete)

Superseded by [AuthenticView](#) and [AuthenticRange](#)

The DocEditView object is renamed to OldAuthenticView.
DocEditSelection is renamed to AuthenticSelection.
DocEditEvent is renamed to AuthenticEvent.
DocEditDataTransfer is renamed to AuthenticDataTransfer.

Their usage - except for AuthenticDataTransfer - is no longer recommended. We will continue to support existing functionality for a yet undefined period of time but no new features will be added to these interface. All functionality available up to now in [DocEditView](#), [DocEditSelection](#), [DocEditEvent](#) and [DocEditDataTransfer](#) is now available via [AuthenticView](#), [AuthenticRange](#) and [AuthenticDataTransfer](#). Many new features have been added.

For examples on migrating from DocEdit to Authentic see the description of the different methods and properties of the different DocEdit objects.

See also

Properties

[altKey](#)
[altLeft](#)
[ctrlKey](#)
[ctrlLeft](#)
[shiftKey](#)
[shiftLeft](#)

[keyCode](#)
[repeat](#)

[button](#)

[clientX](#)
[clientY](#)

[dataTransfer](#)

[srcElement](#)
[fromElement](#)

[propertyName](#)

[cancelBubble](#)
[returnValue](#)

[type](#)

Description

DocEditEvent interface.

altKey (obsolete)

Superseded by parameters to

[AuthenticView.OnKeyboardEvent](#) (On_AuthenticView_KeyPressed)

[AuthenticView.OnMouseEvent](#) (On_AuthenticView_MouseEvent)

[AuthenticView.OnDragOver](#) (On_AuthenticView_DragOver)

The event object that holds the information of the last event is now replaced by parameters to the different event handler functions to simplify data access. The event object will be supported for a not yet defined period of time for compatibility reasons. No improvements are planned. It is highly recommended to migrate to the new event handler functions.

```
// ----- XMLSpy scripting environment - javascript sample -----  
// instead of:  
// function On_DocEditKeyPressed ()  
// {  
//     if ( Application.ActiveDocument.DocEditView.event.altKey ||  
//         Application.ActiveDocument.DocEditView.event.altLeft )  
//         MsgBox ( "alt key is down" );  
// }  
// use now:  
function On_AuthenticView_KeyPressed ( SPYKeyEvent i_eKeyEvent, long  
i_nKeyCode, SPYVirtualKeyMask i_nVirtualKeyStatus )  
{  
    if ( i_nVirtualKeyStatus & spyAltKeyMask )  
        MsgBox ( "alt key is down" );  
}
```

See also

Declaration: `altKey` as Boolean

Description

True if the right ALT key is pressed.

altLeft (obsolete)

Superseded by parameters to

[AuthenticView.OnKeyboardEvent](#) (On_AuthenticView_KeyPressed)

[AuthenticView.OnMouseEvent](#) (On_AuthenticView_MouseEvent)

[AuthenticView.OnDragOver](#) (On_AuthenticView_DragOver)

The event object that holds the information of the last event is now replaced by parameters to the different event handler functions to simplify data access. The event object will be supported for a not yet defined period of time for compatibility reasons. No improvements are planned. It is highly recommended to migrate to the new event handler functions.

```
// ----- XMLSpy scripting environment - javascript sample -----  
// instead of:  
// function On_DocEditKeyDown ()  
// {  
//     if ( Application.ActiveDocument.DocEditView.event.altKey ||  
//         Application.ActiveDocument.DocEditView.event.altLeft)  
//         MsgBox ("alt key is down");  
// }  
// use now:  
function On_AuthenticView_KeyDown (SPYKeyEvent i_eKeyEvent, long i_nKeyCode,  
SPYVirtualKeyMask i_nVirtualKeyStatus)  
{  
    if (i_nVirtualKeyStatus & spyAltKeyMask)  
        MsgBox ("alt key is down");  
}
```

See also

Declaration: `altLeft` as Boolean

Description

True if the left ALT key is pressed.

button (obsolete)**Superseded by parameters to****[AuthenticView.OnMouseEvent](#)** (On_AuthenticView_MouseEvent)**[AuthenticView.OnDragOver](#)** (On_AuthenticView_DragOver)

The event object that holds the information of the last event is now replaced by parameters to the different event handler functions to simplify data access. The event object will be supported for a not yet defined period of time for compatibility reasons. No improvements are planned. It is highly recommended to migrate to the new event handler functions.

```
// ----- XMLSpy scripting environment - javascript sample -----  
// instead of:  
// function On_DocEditButtonDown ()  
// {  
//     if ( Application.ActiveDocument.DocEditView.event.button == 1)  
//         MsgBox ("left mouse button down detected");  
// }  
// use now:  
function On_AuthenticView_MouseEvent ( long i_nXPos, long i_nYPos,  
SPYMouseEvent i_eMouseEvent, IAuthenticRange *i_ipRange)  
{  
    if ( i_eMouseEvent & spyLeftButtonDownMask)  
        MsgBox ("left mouse button down detected");  
}
```

See also**Declaration:** [button](#) as long**Description**

Specifies which mouse button is pressed:

0	No button is pressed.
1	Left button is pressed.
2	Right button is pressed.
3	Left and right buttons are both pressed.
4	Middle button is pressed.
5	Left and middle buttons both are pressed.
6	Right and middle buttons are both pressed.
7	All three buttons are pressed.

cancelBubble (obsolete)

Superseded by the boolean return value of following event handler functions

[AuthenticView.OnKeyboardEvent](#) (On_AuthenticView_KeyPressed)

[AuthenticView.OnMouseEvent](#) (On_AuthenticView_MouseEvent)

[AuthenticView.OnDragOver](#) (On_AuthenticView_DragOver)

The event object that holds the information of the last event is now replaced by parameters to the different event handler functions to simplify data access. The event object will be supported for a not yet defined period of time for compatibility reasons. No improvements are planned. It is highly recommended to migrate to the new event handler functions.

Returning *true* from an event handler function signals that the event has been handled and normal event handling should be aborted.

```
// ----- XMLSpy scripting environment - javascript sample -----
// instead of:
// function On_DocEditKeyPressed ()
// {
//     if ( Application.ActiveDocument.DocEditView.event.keyCode == 0x20)
//     {
//         // cancel key processing, swallow spaces :- )
//         Application.ActiveDocument.DocEditView.event.cancelBubble = true;
//     }
// }
// use now:
function On_AuthenticView_KeyPressed (SPYKeyEvent i_eKeyEvent, long
i_nKeyCode, SPYVirtualKeyMask i_nVirtualKeyStatus)
{
    if ( i_nKeyCode == 0x20)
        return true; // cancel key processing, swallow spaces :- )
}
```

See also

Declaration: [cancelBubble](#) as Boolean

Description

Set [cancelBubble](#) to TRUE if the default event handler should not be called.

clientX (obsolete)**Superseded by parameters to**[AuthenticView.OnMouseEvent](#) (On_AuthenticView_MouseEvent)[AuthenticView.OnBeforeDrop](#) (On_AuthenticView_BeforeDrop)[AuthenticView.OnDragOver](#) (On_AuthenticView_DragOver)

The event object that holds the information of the last event is now replaced by parameters to the different event handler functions to simplify data access. The event object will be supported for a not yet defined period of time for compatibility reasons. No improvements are planned. It is highly recommended to migrate to the new event handler functions.

```
// ----- XMLSpy scripting environment - javascript sample -----
// instead of:
// function On_DocEditMouseMove ()
// {
//     MsgBox ("moving over " +
Application.ActiveDocument.DocEditView.event.clientX +
//         "/" + Application.ActiveDocument.DocEditView.event.clientY);
// }
// use now:
function On_AuthenticView_MouseEvent (long i_nXPos, long i_nYPos,
SPYMouseEvent i_eMouseEvent, IAuthenticRange *i_ipRange)
{
    if (i_eMouseEvent & spyMouseMoveMask)
        MsgBox ("moving over " + i_nXPos + "/" + i_nYPos);
}
```

See also**Declaration:** `clientX` as long**Description**

X value of the current mouse position in client coordinates.

clientY (obsolete)**Superseded by parameters to**

[AuthenticView.OnMouseEvent](#) (On_AuthenticView_MouseEvent)

[AuthenticView.OnBeforeDrop](#) (On_AuthenticView_BeforeDrop)

[AuthenticView.OnDragOver](#) (On_AuthenticView_DragOver)

The event object that holds the information of the last event is now replaced by parameters to the different event handler functions to simplify data access. The event object will be supported for a not yet defined period of time for compatibility reasons. No improvements are planned. It is highly recommended to migrate to the new event handler functions.

```
// ----- XMLSpy scripting environment - javascript sample -----
// instead of:
// function On_DocEditMouseMove ()
// {
//     MsgBox ("moving over " +
Application.ActiveDocument.DocEditView.event.clientX +
//         "/" + Application.ActiveDocument.DocEditView.event.clientY);
// }
// use now:
function On_AuthenticView_MouseEvent (long i_nXPos, long i_nYPos,
SPYMouseEvent i_eMouseEvent, IAuthenticRange *i_ipRange)
{
    if (i_eMouseEvent & spyMouseMoveMask)
        MsgBox ("moving over " + i_nXPos + "/" + i_nYPos);
}
```

See also

Declaration: `clientY` as long

Description

Y value of the current mouse position in client coordinates.

ctrlKey (obsolete)

Superseded by parameters to

[AuthenticView.OnKeyboardEvent](#) (On_AuthenticView_KeyPressed)

[AuthenticView.OnMouseEvent](#) (On_AuthenticView_MouseEvent)

[AuthenticView.OnDragOver](#) (On_AuthenticView_DragOver)

The event object that holds the information of the last event is now replaced by parameters to the different event handler functions to simplify data access. The event object will be supported for a not yet defined period of time for compatibility reasons. No improvements are planned. It is highly recommended to migrate to the new event handler functions.

```
// ----- XMLSpy scripting environment - javascript sample -----  
// instead of:  
// function On_DocEditMouseMove ()  
// {  
//     if ( Application.ActiveDocument.DocEditView.event.ctrlKey ||  
//         Application.ActiveDocument.DocEditView.event.altLeft)  
//         MsgBox ("control key is down");  
// }  
// use now:  
function On_AuthenticView_MouseEvent (long i_nXPos, long i_nYPos,  
SPYMouseEvent i_eMouseEvent, IAuthenticRange *i_ipRange)  
{  
    if (i_eMouseEvent & spyCtrlKeyMask)  
        MsgBox ("control key is down");  
}
```

See also

Declaration: `ctrlKey` as Boolean

Description

True if the right CTRL key is pressed.

ctrlLeft (obsolete)**Superseded by parameters to**[AuthenticView.OnKeyboardEvent](#) (On_AuthenticView_KeyPressed)[AuthenticView.OnMouseEvent](#) (On_AuthenticView_MouseEvent)[AuthenticView.OnDragOver](#) (On_AuthenticView_DragOver)

The event object that holds the information of the last event is now replaced by parameters to the different event handler functions to simplify data access. The event object will be supported for a not yet defined period of time for compatibility reasons. No improvements are planned. It is highly recommended to migrate to the new event handler functions.

```
// ----- XMLSpy scripting environment - javascript sample -----
// instead of:
// function On_DocEditMouseMove ()
// {
//     if ( Application.ActiveDocument.DocEditView.event.ctrlKey ||
//         Application.ActiveDocument.DocEditView.event.altLeft)
//         MsgBox ("control key is down");
// }
// use now:
function On_AuthenticView_MouseEvent (long i_nXPos, long i_nYPos,
SPYMouseEvent i_eMouseEvent, IAuthenticRange *i_ipRange)
{
    if (i_eMouseEvent & spyCtrlKeyMask)
        MsgBox ("control key is down");
}
```

See also**Declaration:** `ctrlLeft` as Boolean**Description**

True if the left CTRL key is pressed.

dataTransfer (obsolete)**Superseded by parameters to****[AuthenticView.OnBeforeDrop](#)** (On_AuthenticView_BeforeDrop)**[AuthenticView.OnDragOver](#)** (On_AuthenticView_DragOver)

The event object that holds the information of the last event is now replaced by parameters to the different event handler functions to simplify data access. The event object will be supported for a not yet defined period of time for compatibility reasons. No improvements are planned. It is highly recommended to migrate to the new event handler functions.

```
// ----- XMLSpy scripting environment - javascript sample -----
// instead of:
// function On_DocEditDrop ()
// {
//     if ( Application.ActiveDocument.DocEditView.event.dataTransfer !=
null)
//         if (!
Application.ActiveDocument.DocEditView.event.dataTransfer.ownDrag)
//         {
//             // cancel key processing, don't drop foreign objects :-)
//             Application.ActiveDocument.DocEditView.event.cancelBubble =
true;
//         }
// }
// use now:
function On_AuthenticView_BeforeDrop ( long i_nXPos, long i_nYPos,
IAuthenticRange *i_ipRange,
IAuthenticDataTransfer *i_ipData)
{
    if (i_ipRange != null)
        if (! i_ipRange.ownDrag)
            return true; // cancel key processing, don't drop foreign
objects :-)
    return false;
}
```

See also**Declaration:** [dataTransfer](#) as Variant**Description**Property [dataTransfer](#)

fromElement (obsolete)**Not supported****See also****Declaration:** `fromElement` as Variant (not supported)**Description**

Currently no event sets this property.

keyCode (obsolete)**Superseded by a parameter to [AuthenticView.OnKeyboardEvent](#) (On_AuthenticView_KeyPressed)**

The event object that holds the information of the last event is now replaced by parameters to the different event handler functions to simplify data access. The event object will be supported for a not yet defined period of time for compatibility reasons. No improvements are planned. It is highly recommended to migrate to the new event handler functions.

```
// ----- XMLSpy scripting environment - javascript sample -----  
// instead of:  
// function On_DocEditKeyPressed ()  
// {  
//     if ( Application.ActiveDocument.DocEditView.event.keyCode == 0x20)  
//     {  
//         // cancel key processing, swallow spaces :-)  
//         Application.ActiveDocument.DocEditView.event.cancelBubble = true;  
//     }  
// }  
// use now:  
function On_AuthenticView_KeyPressed (SPYKeyEvent i_eKeyEvent, long  
i_nKeyCode, SPYVirtualKeyMask i_nVirtualKeyStatus)  
{  
    if ( i_nKeyCode == 0x20)  
        return true; // cancel key processing, swallow spaces :-)  
}
```

See also**Declaration:** `keyCode` as long**Description**

Keycode of the currently pressed key. This property is read-write.

propertyName (obsolete)

Not supported

See also

Declaration: `propertyName` as String (not supported)

Description

Currently no event sets this property.

repeat (obsolete)

Not supported

See also

Declaration: `repeat` as Boolean (not supported)

Description

True if the `onkeydown` event is repeated.

returnValue (obsolete)

No longer supported

See also

Declaration: `returnValue` as Variant

Description

Use `returnValue` to set a return value for your event handler.

shiftKey (obsolete)**Superseded by parameters to**[AuthenticView.OnKeyboardEvent](#) (On_AuthenticView_KeyPressed)[AuthenticView.OnMouseEvent](#) (On_AuthenticView_MouseEvent)[AuthenticView.OnDragOver](#) (On_AuthenticView_DragOver)

The event object that holds the information of the last event is now replaced by parameters to the different event handler functions to simplify data access. The event object will be supported for a not yet defined period of time for compatibility reasons. No improvements are planned. It is highly recommended to migrate to the new event handler functions.

```
// ----- XMLSpy scripting environment - javascript sample -----
// instead of:
// function On_DocEditDragOver ()
// {
//     if ( Application.ActiveDocument.DocEditView.event.shiftKey ||
//         Application.ActiveDocument.DocEditView.event.shiftLeft)
//         MsgBox ("shift key is down");
// }
// use now:
function On_AuthenticView_DragOver (long i_nXPos, long i_nYPos,
                                   SPYMouseEvent i_eMouseEvent,
                                   IAuthenticRange *i_ipRange,
                                   IAuthenticDataTransfer *i_ipData)
{
    if (i_eMouseEvent & spyShiftKeyMask)
        MsgBox ("shift key is down");
}
```

See also**Declaration:** [shiftKey](#) as Boolean**Description**

True if the right SHIFT key is pressed.

shiftLeft (obsolete)

Superseded by parameters to

[AuthenticView.OnKeyboardEvent](#) (On_AuthenticView_KeyPressed)

[AuthenticView.OnMouseEvent](#) (On_AuthenticView_MouseEvent)

[AuthenticView.OnDragOver](#) (On_AuthenticView_DragOver)

The event object that holds the information of the last event is now replaced by parameters to the different event handler functions to simplify data access. The event object will be supported for a not yet defined period of time for compatibility reasons. No improvements are planned. It is highly recommended to migrate to the new event handler functions.

```
// ----- XMLSpy scripting environment - javascript sample -----
// instead of:
// function On_DocEditDragOver ( )
// {
//     if ( Application.ActiveDocument.DocEditView.event.shiftKey ||
//         Application.ActiveDocument.DocEditView.event.shiftLeft)
//         MsgBox ("shift key is down");
// }
// use now:
function On_AuthenticView_DragOver (long i_nXPos, long i_nYPos,
                                   SPYMouseEvent i_eMouseEvent,
                                   IAuthenticRange *i_ipRange,
                                   IAuthenticDataTransfer *i_ipData)
{
    if (i_eMouseEvent & spyShiftKeyMask)
        MsgBox ("shift key is down");
}
```

See also

Declaration: [shiftLeft](#) as Boolean

Description

True if the left SHIFT key is pressed.

srcElement (obsolete)**Superseded by parameters to****[AuthenticView.OnMouseEvent](#)** (On_AuthenticView_MouseEvent)**[AuthenticView.OnBeforeDrop](#)** (On_AuthenticView_BeforeDrop)**[AuthenticView.OnDragOver](#)** (On_AuthenticView_DragOver)

The event object that holds the information of the last event is now replaced by parameters to the different event handler functions to simplify data access. The event object will be supported for a not yet defined period of time for compatibility reasons. No improvements are planned. It is highly recommended to migrate to the new event handler functions.

With the new event handler function, a range object selecting this element is provided instead of the XMLData element currently below the mouse cursor.

```
// ----- XMLSpy scripting environment - javascript sample -----
// instead of:
// function On_DocEditMouseMove ()
// {
//     var objEvent = Application.ActiveDocument.DocEditView.event;
//     if (objEvent.srcElement != null)
//         MsgBox ("moving over " + objEvent.srcElement.Parent.Name);
// }
// use now:
function On_AuthenticView_MouseEvent (long i_nXPos, long i_nYPos,
SPYMouseEvent i_eMouseEvent, IAuthenticRange *i_ipRange)
{
    if ((i_eMouseEvent & spyMouseMoveMask) &&
        (i_ipRange != null))
        MsgBox ("moving over " + i_ipRange.FirstXMLData.Parent.Name);
}
```

See also**Declaration:** [srcElement](#) as Variant**Description**

Element which fires the current event. This is usually an [XMLData](#) object.

type (obsolete)

Not supported

See also

Declaration: `type` as String (not supported)

Description

Currently no event sets this property.

3.3.2 AuthenticSelection (obsolete)

Superseded by [AuthenticRange](#)

The DocEditView object is renamed to OldAuthenticView.
DocEditSelection is renamed to AuthenticSelection.
DocEditEvent is renamed to AuthenticEvent.
DocEditDataTransfer is renamed to AuthenticDataTransfer.

Their usage - except for AuthenticDataTransfer - is no longer recommended. We will continue to support existing functionality for a yet undefined period of time but no new features will be added to these interface. All functionality available up to now in [DocEditView](#), [DocEditSelection](#), [DocEditEvent](#) and [DocEditDataTransfer](#) is now available via [AuthenticView](#), [AuthenticRange](#) and [AuthenticDataTransfer](#). Many new features have been added.

For examples on migrating from DocEdit to Authentic see the description of the different methods and properties of the different DocEdit objects.

See also**Properties**

[Start](#)
[StartTextPosition](#)
[End](#)
[EndTextPosition](#)

End (obsolete)

Superseded by [AuthenticRange.LastXMLData](#)

```
// ----- javascript sample -----  
// instead of:  
// var objXMLData =  
Application.ActiveDocument.DocEditView.CurrentSelection.End;  
// use now:  
var objXMLData =  
Application.ActiveDocument.AuthenticView.Selection.LastXMLData;
```

See also

Declaration: End as [XMLData](#)

Description

XML element where the current selection ends.

EndTextPosition (obsolete)

Superseded by [AuthenticRange.LastXMLDataOffset](#)

```
// ----- javascript sample -----  
// instead of:  
// var nOffset =  
Application.ActiveDocument.DocEditView.CurrentSelection.EndTextPosition;  
// use now:  
var nOffset =  
Application.ActiveDocument.AuthenticView.Selection.LastXMLDataOffset;
```

See also

Declaration: EndTextPosition as long

Description

Position in [DocEditSelection.End.TextValue](#) where the selection ends.

Start (obsolete)**Superseded by** [AuthenticRange.FirstXMLData](#)

```
// ----- javascript sample -----  
// instead of:  
// var objXMLData =  
Application.ActiveDocument.DocEditView.CurrentSelection.Start;  
// use now:  
var objXMLData =  
Application.ActiveDocument.AuthenticView.Selection.FirstXMLData;
```

See also**Declaration:** `Start` as [XMLData](#)**Description**

XML element where the current selection starts.

StartTextPosition (obsolete)**Superseded by** [AuthenticRange.FirstXMLDataOffset](#)

```
// ----- javascript sample -----  
// instead of:  
// var nOffset =  
Application.ActiveDocument.DocEditView.CurrentSelection.StartTextPosition;  
// use now:  
var nOffset =  
Application.ActiveDocument.AuthenticView.Selection.FirstXMLDataOffset;
```

See also**Declaration:** `StartTextPosition` as long**Description**Position in [DocEditSelection.Start.TextValue](#) where the selection starts.

3.3.3 OldAuthenticView (obsolete)

Superseded by [AuthenticView](#) and [AuthenticRange](#)

The DocEditView object is renamed to OldAuthenticView.
DocEditSelection is renamed to AuthenticSelection.
DocEditEvent is renamed to AuthenticEvent.
DocEditDataTransfer is renamed to AuthenticDataTransfer.

Their usage - except for AuthenticDataTransfer - is no longer recommended. We will continue to support existing functionality for a yet undefined period of time but no new features will be added to these interfaces. All functionality available up to now in [DocEditView](#), [DocEditSelection](#), [DocEditEvent](#) and [DocEditDataTransfer](#) is now available via [AuthenticView](#), [AuthenticRange](#) and [AuthenticDataTransfer](#). Many new features have been added.

For examples on migrating from DocEdit to Authentic see the description of the different methods and properties of the different DocEdit objects.

See also**Methods**

[LoadXML](#)

[SaveXML](#)

[EditClear](#)

[EditCopy](#)

[EditCut](#)

[EditPaste](#)

[EditRedo](#)

[EditSelectAll](#)

[EditUndo](#)

[RowAppend](#)

[RowDelete](#)

[RowDuplicate](#)

[RowInsert](#)

[RowMoveDown](#)

[RowMoveUp](#)

[ApplyTextState](#)

[IsTextStateApplied](#)

[IsTextStateEnabled](#)

[MarkUpView](#)

[SelectionSet](#)

[SelectionMoveTabOrder](#)

[GetNextVisible](#)

[GetPreviousVisible](#)

[GetAllowedElements](#)

Properties[CurrentSelection](#)[event](#)[XMLRoot](#)[IsEditClearEnabled](#)[IsEditCopyEnabled](#)[IsEditCutEnabled](#)[IsEditPasteEnabled](#)[IsEditRedoEnabled](#)[IsEditUndoEnabled](#)[IsRowAppendEnabled](#)[IsRowDeleteEnabled](#)[IsRowDuplicateEnabled](#)[IsRowInsertEnabled](#)[IsRowMoveDownEnabled](#)[IsRowMoveUpEnabled](#)**Description**

Interface for Authentic View.

ApplyTextState (obsolete)

Superseded by [AuthenticRange.PerformAction](#)

Use spyAuthenticApply for the eAction parameter. The PerformAction method allows to apply text state attributes to any range of the document, not only the current UI selection.

```
// ----- javascript sample -----  
// instead of:  
// Application.ActiveDocument.DocEditView.ApplyTextState ("bold");  
// use now:  
if (! Application.ActiveDocument.AuthenticView.Selection.PerformAction  
(spyAuthenticApply, "bold"))  
    MsgBox ("Error: can't set current selection to bold");
```

See also

Declaration: [ApplyTextState](#) (*elementName* as String)

Description

Applies or removes the text state defined by the parameter `elementName`. Common examples for the parameter `elementName` would be `strong` and `italic`.

In an XML document there are segments of data, which may contain sub-elements. For example consider the following HTML:

```
<b>fragment</b>
```

The HTML tag `<b>` will cause the word `fragment` to be bold. However, this only happens because the HTML parser knows that the tag `<b>` is bold. With XML there is much more flexibility. It is possible to define any XML tag to do anything you desire. The point is that it is possible to apply a Text state using XML. But the Text state that is applied must be part of the schema. For example in the `OrgChart.xml`, `OrgChart.sps`, `OrgChart.xsd` example the tag `<strong>` is the same as bold. And to apply bold the method `ApplyTextState()` is called. But like the row and edit operations it is necessary to test if it is possible to apply the text state.

See also [IsTextStateEnabled](#) and [IsTextStateApplied](#).

CurrentSelection (obsolete)**Superseded by [AuthenticView.Selection](#)**

The returned [AuthenticRange](#) object supports navigation via XMLData elements as well as navigation by document elements (e.g. characters, words, tags) or text cursor positions.

```
// ----- javascript sample -----  
// instead of:  
// var objDocEditSel =  
Application.ActiveDocument.DocEditView.CurrentSelection;  
// use now:  
var objRange = Application.ActiveDocument.AuthenticView.Selection;
```

See also

Declaration: [CurrentSelection](#) as [DocEditSelection](#)

Description

The property provides access to the current selection in the Authentic View.

EditClear (obsolete)**Superseded by [AuthenticRange.Delete](#)**

The Delete method of AuthenticRange allows to delete any range of the document, not only the current UI selection.

```
// ----- javascript sample -----  
// instead of:  
// Application.ActiveDocument.DocEditView.EditClear();  
// use now:  
if (! Application.ActiveDocument.AuthenticView.Selection.Delete())  
    MsgBox ("Error: can't delete current selection");
```

See also

Declaration: [EditClear](#)

Description

Deletes the current selection.

EditCopy (obsolete)**Superseded by [AuthenticRange.Copy](#)**

The Copy method of AuthenticRange allows to delete any range of the document, not only the current UI selection.

```
// ----- javascript sample -----  
// instead of:  
// Application.ActiveDocument.DocEditView.EditCopy();  
// use now:  
if (! Application.ActiveDocument.AuthenticView.Selection.Copy())  
    MsgBox ("Error: can't copy current selection");
```

See also

Declaration: [EditCopy](#)

Description

Copies the current selection to the clipboard.

EditCut (obsolete)**Superseded by [AuthenticRange.Cut](#)**

The Cut method of AuthenticRange allows to delete any range of the document, not only the current UI selection.

```
// ----- javascript sample -----  
// instead of:  
// Application.ActiveDocument.DocEditView.EditCut();  
// use now:  
if (! Application.ActiveDocument.AuthenticView.Selection.Cut())  
    MsgBox ("Error: can't cut out current selection");
```

See also

Declaration: [EditCut](#)

Description

Cuts the current selection from the document and copies it to the clipboard.

EditPaste (obsolete)**Superseded by [AuthenticRange.Paste](#)**

The Paste method of AuthenticRange allows to delete any range of the document, not only the current UI selection.

```
// ----- javascript sample -----  
// instead of:  
// Application.ActiveDocument.DocEditView.EditPaste();  
// use now:  
if (! Application.ActiveDocument.AuthenticView.Selection.Paste())  
    MsgBox ("Error: can't paste to current selection");
```

See also

Declaration: [EditPaste](#)

Description

Pastes the content from the clipboard into the document.

EditRedo (obsolete)**Superseded by [AuthenticView.Redo](#)**

```
// ----- javascript sample -----  
// instead of:  
// Application.ActiveDocument.DocEditView.EditRedo();  
// use now:  
if (! Application.ActiveDocument.AuthenticView.Redo())  
    MsgBox ("Error: no redo step available");
```

See also

Declaration: [EditRedo](#)

Description

Redo the last undo step.

EditSelectAll (obsolete)

Superseded by [AuthenticView.WholeDocument](#) and [AuthenticRange.Select](#)

```
// ----- javascript sample -----  
// instead of:  
// Application.ActiveDocument.DocEditView.EditSelectAll();  
// use now:  
Application.ActiveDocument.AuthenticView.WholeDocument.Select();
```

See also

Declaration: [EditSelectAll](#)

Description

The method selects the complete document.

EditUndo (obsolete)

Superseded by [AuthenticView.Undo](#)

```
// ----- javascript sample -----  
// instead of:  
// Application.ActiveDocument.DocEditView.EditUndo();  
// use now:  
if (! Application.ActiveDocument.AuthenticView.Undo())  
    MsgBox ("Error: no undo step available");
```

See also

Declaration: [EditUndo](#)

Description

Undo the last action.

event (obsolete)

Superseded by parameters to [AuthenticView events](#).

See also

Declaration: `event` as [DocEditEvent](#)

Description

The event property holds a `DocEditEvent` object which contains information about the current event.

GetAllowedElements (obsolete)

Superseded by [AuthenticRange.CanPerformActionWith](#)

`AuthenticRange` now supports all functionality of the 'elements' entry helper. Besides querying the elements that can be inserted, appended, etc., you can invoke the action as well. See [AuthenticRange.PerformAction](#) for more information.

```
// ----- javascript sample -----  
// instead of:  
// var arrElements = New Array();  
// var objDocEditView = Application.ActiveDocument.DocEditView;  
// var objStartElement = objDocEditView.CurrentSelection.Start;  
// var objEndElement = objDocEditView.CurrentSelection.End;  
// objDocEditView.GetAllowedElements(k_ActionInsertBefore, objStartElement,  
objEndElement, arrElements);  
// use now:  
var arrElements = New Array();  
Application.ActiveDocument.AuthenticView.Selection.CanPerformActionWith  
(spyAuthenticInsertBefore, arrElements);
```

See also

Declaration: `GetAllowedElements` (*nAction* as [SpyAuthenticElementActions](#),
pStartElement as [XMLData](#), *pEndElement* as [XMLData](#), *pElements* as Variant)

Description

`GetAllowedElements()` returns the allowed elements for the various actions specified by *nAction*.

JavaScript example:

```
Function GetAllowed()  
{  
    var objView = Application.ActiveDocument.DocEditView;  
  
    var arrElements = New Array(1);
```



```

var objStart = objView.CurrentSelection.Start;
var objEnd = objView.CurrentSelection.End;

var strText;
strText = "valid elements at current selection:\n\n";

For( var i = 1; i <= 4; i++) {
    objPlugIn.GetAllowedElements( i, objStart, objEnd, arrElements);
    strText = strText + ListArray( arrElements) + "-----\n";
}

Return strText;
}

Function ListArray( arrIn)
{
    var strText = "";

    If( TypeOf( arrIn) == "object") {
        For( var i = 0; i <= ( arrIn.length - 1); i++)
            strText = strText + arrIn[i] + "\n";
    }

    Return strText;
}

```

VBScript example:

```

Sub DisplayAllowed
    Dim objView
    Set objView = Application.ActiveDocument.DocEditView

    Dim arrElements()

    Dim objStart
    Dim objEnd
    Set objStart = objView.CurrentSelection.Start
    Set objEnd = objView.CurrentSelection.End

    Dim strText
    strText = "valid elements at current selection:" & chr(13) & chr(13)

    Dim i

    For i = 1 To 4
        objView.GetAllowedElements i, objStart, objEnd, arrElements
        strText = strText & ListArray( arrElements) & "-----" & chr(13)
    Next

    msgbox strText
End Sub

Function ListArray( arrIn)
    Dim strText

    If IsArray( arrIn) Then
        Dim i

        For i = 0 To UBound( arrIn)
            strText = strText & arrIn(i) & chr(13)
        Next
    End If

    ListArray = strText
End Function

```


GetNextVisible (obsolete)**Superseded by [AuthenticRange.SelectNext](#)**

AuthenticRange now supports a wide range of element navigation methods based on document elements like characters, words, tags and many more. Selecting the text passage that represents the content of the next XML element is just one of them.

```
// ----- javascript sample -----  
// instead of:  
// var objCurrXMLData = ...  
// var objXMLData =  
Application.ActiveDocument.DocEditView.GetNextVisible(objCurrXMLData);  
// Application.ActiveDocument.DocEditView.SelectionSet (objXMLData, 0,  
objXMLData, -1);  
// use now:  
var objRange = ...  
try  
    { objRange.SelectNext (spyAuthenticTag).Select(); }  
catch (err)  
{  
    if ((err.number & 0xffff) == 2003)  
        MsgBox ("end of document reached");  
    else  
        throw (err);  
}
```

See also

Declaration: `GetNextVisible (pElement as XMLData) as XMLData`

Description

The method gets the next visible XML element in the document.

GetPreviousVisible (obsolete)**Superseded by [AuthenticRange.SelectPrevious](#)**

AuthenticRange now supports a wide range of element navigation methods based on document elements like characters, words, tags and many more. Selecting the text passage that represents the content of the previous XML element is just one of them.

```
// ----- javascript sample -----  
// instead of:  
// var objCurrXMLData = ...  
// var objXMLData =  
Application.ActiveDocument.DocEditView.GetPreviousVisible(objCurrXMLData);  
// Application.ActiveDocument.DocEditView.SelectionSet (objXMLData, 0,  
objXMLData, -1);  
// use now:  
var objRange = ...  
try  
    { objRange.SelectPrevious (spyAuthenticTag).Select(); }  
catch (err)  
{  
    if ((err.number & 0xffff) == 2004)  
        MsgBox ("begin of document reached");  
    else  
        throw (err);  
}
```

See also

Declaration: `GetPreviousVisible (pElement as XMLData) as XMLData`

Description

The method gets the previous visible XML element in the document.

IsEditClearEnabled (obsolete)**Superseded by [AuthenticRange.IsDeleteEnabled](#)**

The IsDeleteEnabled property is now supported for any range of the document, not only the current UI selection.

```
// ----- javascript sample -----  
// instead of:  
// if ( Application.ActiveDocument.DocEditView.IsEditClearEnabled)  
//     Application.ActiveDocument.DocEditView.EditClear();  
// use now:  
var objCurrSelection = Application.ActiveDocument.AuthenticView.Selection;  
if ( objCurrSelection.IsDeleteEnabled)  
    objCurrSelection.Delete();
```

See also

Declaration: [IsEditClearEnabled](#) as Boolean

Description

True if [EditClear](#) is possible. See also [Editing operations](#).

IsEditCopyEnabled (obsolete)**Superseded by [AuthenticRange.IsCopyEnabled](#)**

The IsCopyEnabled property is now supported for any range of the document, not only the current UI selection.

```
// ----- javascript sample -----  
// instead of:  
// if ( Application.ActiveDocument.DocEditView.IsEditCopyEnabled)  
//     Application.ActiveDocument.DocEditView.EditCopy();  
// use now:  
var objCurrSelection = Application.ActiveDocument.AuthenticView.Selection;  
if ( objCurrSelection.IsCopyEnabled)  
    objCurrSelection.Copy();
```

See also

Declaration: [IsEditCopyEnabled](#) as Boolean

Description

True if copy to clipboard is possible. See also [EditCopy](#) and [Editing operations](#).

IsEditCutEnabled (obsolete)**Superseded by [AuthenticRange.IsCutEnabled](#)**

The IsCutEnabled property is now supported for any range of the document, not only the current UI selection.

```
// ----- javascript sample -----  
// instead of:  
// if ( Application.ActiveDocument.DocEditView.IsEditCutEnabled)  
//     Application.ActiveDocument.DocEditView.EditCut();  
// use now:  
var objCurrSelection = Application.ActiveDocument.AuthenticView.Selection;  
if ( objCurrSelection.IsCutEnabled)  
    objCurrSelection.Cut();
```

See also

Declaration: [IsEditCutEnabled](#) as Boolean

Description

True if [EditCut](#) is currently possible. See also [Editing operations](#).

IsEditPasteEnabled (obsolete)**Superseded by [AuthenticRange.IsPasteEnabled](#)**

The IsPasteEnabled property is now supported for any range of the document, not only the current UI selection.

```
// ----- javascript sample -----  
// instead of:  
// if ( Application.ActiveDocument.DocEditView.IsEditPasteEnabled)  
//     Application.ActiveDocument.DocEditView.EditPaste();  
// use now:  
var objCurrSelection = Application.ActiveDocument.AuthenticView.Selection;  
if ( objCurrSelection.IsPasteEnabled)  
    objCurrSelection.Paste();
```

See also

Declaration: [IsEditPasteEnabled](#) as Boolean

Description

True if [EditPaste](#) is possible. See also [Editing operations](#).

IsEditRedoEnabled (obsolete)**Superseded by [AuthenticView.IsRedoEnabled](#)**

```
// ----- javascript sample -----  
// instead of:  
// if ( Application.ActiveDocument.DocEditView.IsEditRedoEnabled)  
//     Application.ActiveDocument.DocEditView.EditRedo();  
// use now:  
if ( Application.ActiveDocument.AuthenticView.IsRedoEnabled)  
    Application.ActiveDocument.AuthenticView.Redo();
```

See also**Declaration:** [IsEditRedoEnabled](#) as Boolean**Description**True if [EditRedo](#) is currently possible. See also [Editing operations](#).**IsEditUndoEnabled (obsolete)****Superseded by [AuthenticView.IsUndoEnabled](#)**

```
// ----- javascript sample -----  
// instead of:  
// if ( Application.ActiveDocument.DocEditView.IsEditUndoEnabled)  
//     Application.ActiveDocument.DocEditView.EditUndo();  
// use now:  
if ( Application.ActiveDocument.AuthenticView.IsUndoEnabled)  
    Application.ActiveDocument.AuthenticView.Undo();
```

See also**Declaration:** [IsEditUndoEnabled](#) as Boolean**Description**True if [EditUndo](#) is possible. See also [Editing operations](#).

IsRowAppendEnabled (obsolete)**Superseded by [AuthenticRange.IsInDynamicTable](#)**

The operations 'insert', 'append', 'delete' and 'duplicate' row are available whenever the selection is inside a dynamic table.

```
// ----- javascript sample -----  
// instead of:  
// if ( Application.ActiveDocument.DocEditView.IsRowAppendEnabled)  
//     Application.ActiveDocument.DocEditView.RowAppend();  
// use now:  
if ( Application.ActiveDocument.AuthenticView.Selection.IsInDynamicTable()  
    Application.ActiveDocument.AuthenticView.Selection.AppendRow();
```

See also

Declaration: [IsRowAppendEnabled](#) as Boolean

Description

True if [RowAppend](#) is possible. See also [Row operations](#).

IsRowDeleteEnabled (obsolete)**Superseded by [AuthenticRange.IsInDynamicTable](#)**

The operations 'insert', 'append', 'delete' and 'duplicate' row are available whenever the selection is inside a dynamic table.

```
// ----- javascript sample -----  
// instead of:  
// if ( Application.ActiveDocument.DocEditView.IsRowDeleteEnabled)  
//     Application.ActiveDocument.DocEditView.Rowdelete();  
// use now:  
if ( Application.ActiveDocument.AuthenticView.Selection.IsInDynamicTable()  
    Application.ActiveDocument.AuthenticView.Selection.DeleteRow();
```

See also

Declaration: [IsRowDeleteEnabled](#) as Boolean

Description

True if [RowDelete](#) is possible. See also [Row operations](#).

IsRowDuplicateEnabled (obsolete)**Superseded by [AuthenticRange.IsInDynamicTable](#)**

The operations 'insert', 'append', 'delete' and 'duplicate' row are available whenever the selection is inside a dynamic table.

```
// ----- javascript sample -----  
// instead of:  
// if ( Application.ActiveDocument.DocEditView.IsRowDuplicateEnabled)  
//     Application.ActiveDocument.DocEditView.RowDuplicate( );  
// use now:  
if ( Application.ActiveDocument.AuthenticView.Selection.IsInDynamicTable()  
    Application.ActiveDocument.AuthenticView.Selection.DuplicateRow( );
```

See also

Declaration: [IsRowDuplicateEnabled](#) as Boolean

Description

True if [RowDuplicate](#) is currently possible. See also [Row operations](#).

IsRowInsertEnabled (obsolete)**Superseded by [AuthenticRange.IsInDynamicTable](#)**

The operations 'insert', 'append', 'delete' and 'duplicate' row are available whenever the selection is inside a dynamic table.

```
// ----- javascript sample -----  
// instead of:  
// if ( Application.ActiveDocument.DocEditView.IsRowInsertEnabled)  
//     Application.ActiveDocument.DocEditView.RowInsert( );  
// use now:  
if ( Application.ActiveDocument.AuthenticView.Selection.IsInDynamicTable()  
    Application.ActiveDocument.AuthenticView.Selection.InsertRow( );
```

See also

Declaration: [IsRowInsertEnabled](#) as Boolean

Description

True if [RowInsert](#) is possible. See also [Row operations](#).

IsRowMoveDownEnabled (obsolete)**Superseded by [AuthenticRange.IsLastRow](#)**

```
// ----- javascript sample -----  
// instead of:  
// if ( Application.ActiveDocument.OldAuthenticView.IsRowMoveDownEnabled)  
//     Application.ActiveDocument.DocEditView.RowMoveDown( );  
// use now:  
if ( ! Application.ActiveDocument.AuthenticView.Selection.IsLastRow)  
    Application.ActiveDocument.AuthenticView.Selection.MoveRowDown( );
```

See also**Declaration:** [IsRowMoveDownEnabled](#) as Boolean**Description**True if [RowMoveDown](#) is currently possible. See also [Row operations](#).**IsRowMoveUpEnabled (obsolete)****Superseded by [AuthenticRange.IsFirstRow](#)**

```
// ----- javascript sample -----  
// instead of:  
// if ( Application.ActiveDocument.DocEditView.IsRowMoveUpEnabled)  
//     Application.ActiveDocument.DocEditView.RowMoveUp( );  
// use now:  
if ( ! Application.ActiveDocument.AuthenticView.Selection.IsFirstRow)  
    Application.ActiveDocument.AuthenticView.Selection.MoveRowUp( );
```

See also**Declaration:** [IsRowMoveUpEnabled](#) as Boolean**Description**True if [RowMoveUp](#) is possible. See also [Row operations](#).

IsTextStateApplied (obsolete)

Superseded by [AuthenticRange.IsTextStateApplied](#)

```
// ----- javascript sample -----  
// instead of:  
// Application.ActiveDocument.DocEditView.IsTextStateApplied ("bold");  
// use now:  
if ( Application.ActiveDocument.AuthenticView.Selection.IsTextStateApplied (  
"bold"))  
    MsgBox ("bold on");  
else  
    MsgBox ("bold off");
```

See also

Declaration: `IsTextStateApplied` (*elementName* as String) as Boolean

Description

Checks to see if the it the text state has already been applied. Common examples for the parameter `elementName` would be strong and italic.

IsTextStateEnabled (obsolete)

Superseded by [AuthenticRange.CanPerformAction](#)

Use `spyAuthenticApply` for the `eAction` parameter. The `CanPerformAction` method allows to operate on any range of the document, not only the current UI selection.

```
// ----- javascript sample -----  
// instead of:  
// Application.ActiveDocument.DocEditView.IsTextStateEnabled ("bold");  
// use now:  
if ( Application.ActiveDocument.AuthenticView.Selection.CanPerformAction  
(spyAuthenticApply, "bold"))  
    ... // e.g. enable 'bold' button
```

See also

Declaration: `IsTextStateEnabled` (*i_strElementName* as String) as Boolean

Description

Checks to see if it is possible to apply a text state. Common examples for the parameter `elementName` would be strong and italic.

LoadXML (obsolete)**Superseded by [AuthenticView.AsXMLString](#)**

AuthenticView now supports the property AsXMLString that can be used to directly access and replace the document content as an XMLString.

```
// ----- javascript sample -----  
// instead of:  
// Application.ActiveDocument.DocEditView.LoadXML ( strDocAsXMLString );  
// use now:  
try  
{ Application.ActiveDocument.AuthenticView.AsXMLString =  
strDocAsXMLString; }  
catch (err)  
{ MsgBox ("Error: invalid XML string"); }
```

See also

Declaration: `LoadXML` (*xmlString* as String)

Description

Loads the current XML document with the XML string applied. The new content is displayed immediately.

The *xmlString* parameter must begin with the XML declaration, e.g.,
`objPlugIn.LoadXML("<?xml version='1.0' encoding='UTF-8'?><root></root>");`

MarkUpView (obsolete)**Superseded by [AuthenticView.MarkupVisibility](#)**

```
// ----- javascript sample -----  
// instead of:  
// Application.ActiveDocument.DocEditView.MarkuUpView = 2;  
// use now:  
Application.ActiveDocument.AuthenticView.MarkupVisibility =  
spyAuthenticMarkupLarge;
```

See also

Declaration: `MarkUpView` (*kind* as long)

Description

By default the document displayed is using HTML techniques. But sometimes it is desirable to show the editing tags. Using this method it is possible to display three different types of markup tags:

- 0 hide the markup tags
- 2 show the large markup tags
- 3 show the mixed markup tags.

RowAppend (obsolete)**Superseded by [AuthenticRange.AppendRow](#)**

The table operations of AuthenticRange now allow to manipulate any table in the current document independent of the current UI selection.

```
// ----- javascript sample -----  
// instead of:  
// Application.ActiveDocument.DocEditView.RowAppend();  
// use now:  
if (! Application.ActiveDocument.AuthenticView.Selection.AppendRow())  
    MsgBox ("Error: can't append row");
```

See also

Declaration: [RowAppend](#)

Description

Appends a row at the current position.

See also [Row operations](#).

RowDelete (obsolete)**Superseded by [AuthenticRange.DeleteRow](#)**

The table operations of AuthenticRange now allow to manipulate any table in the current document independent of the current UI selection.

```
// ----- javascript sample -----  
// instead of:  
// Application.ActiveDocument.DocEditView.RowDelete();  
// use now:  
if (! Application.ActiveDocument.AuthenticView.Selection.DeleteRow())  
    MsgBox ("Error: can't delete row");
```

See also

Declaration: [RowDelete](#)

Description

Deletes the currently selected row(s).

See also [Row operations](#).

RowDuplicate (obsolete)

Superseded by [AuthenticRange.DuplicateRow](#)

The table operations of AuthenticRange now allow to manipulate any table in the current document independent of the current UI selection.

```
// ----- javascript sample -----  
// instead of:  
// Application.ActiveDocument.DocEditView.RowDuplicate();  
// use now:  
if (! Application.ActiveDocument.AuthenticView.Selection.DuplicateRow())  
    MsgBox ("Error: can't duplicate row");
```

See also

Declaration: [RowDuplicate](#)

Description

The method duplicates the currently selected rows.

See also [Row operations](#).

RowInsert (obsolete)

Superseded by [AuthenticRange.InsertRow](#)

The table operations of AuthenticRange now allow to manipulate any table in the current document independent of the current UI selection.

```
// ----- javascript sample -----  
// instead of:  
// Application.ActiveDocument.DocEditView.RowInsert();  
// use now:  
if (! Application.ActiveDocument.AuthenticView.Selection.InsertRow())  
    MsgBox ("Error: can't insert row");
```

See also

Declaration: [RowInsert](#)

Description

Inserts a new row immediately above the current selection.

See also [Row operations](#).

RowMoveDown (obsolete)**Superseded by [AuthenticRange.MoveRowDown](#)**

The table operations of AuthenticRange now allow to manipulate any table in the current document independent of the current UI selection.

```
// ----- javascript sample -----  
// instead of:  
// Application.ActiveDocument.DocEditView.RowMoveDown();  
// use now:  
if (! Application.ActiveDocument.AuthenticView.Selection.MoveRowDown())  
    MsgBox ("Error: can't move row down");
```

See also

Declaration: [RowMoveDown](#)

Description

Moves the current row one position down.

See also [Row operations](#).

RowMoveUp (obsolete)**Superseded by [AuthenticRange.MoveRowUp](#)**

The table operations of AuthenticRange now allow to manipulate any table in the current document independent of the current UI selection.

```
// ----- javascript sample -----  
// instead of:  
// Application.ActiveDocument.DocEditView.RowAppend();  
// use now:  
if (! Application.ActiveDocument.AuthenticView.Selection.MoveRowUp())  
    MsgBox ("Error: can't move row up");
```

See also

Declaration: [RowMoveUp](#)

Description

Moves the current row one position up.

See also [Row operations](#).

SaveXML (obsolete)**Superseded by [AuthenticView.AsXMLString](#)**

AuthenticView now supports the property XMLString that can be used to directly access and replace the document content as an XMLString.

```
// ----- javascript sample -----  
// instead of:  
// var strDocAsXMLString = Application.ActiveDocument.DocEditView.SaveXML();  
// use now:  
try  
{  
    var strDocAsXMLString =  
Application.ActiveDocument.AuthenticView.AsXMLString;  
    ... // do something here  
}  
catch (err)  
{ MsgBox ("Error: invalid XML string"); }
```

See also

Declaration: [SaveXML](#) as String

Return Value

XML structure as string

Description

Saves the current XML data to a string that is returned to the caller.

SelectionMoveTabOrder (obsolete)**Superseded by [AuthenticRange.SelectNext](#)**

AuthenticRange now supports a wide range of element navigation methods based on document elements like characters, words, tags and many more. Selecting the next paragraph is just one of them, and navigation is not necessarily bound to the current UI selection.

```
// ----- javascript sample -----  
// instead of:  
// Application.ActiveDocument.DocEditView.SelectionMoveTabOrder(true, true);  
// use now:  
Application.ActiveDocument.AuthenticView.Selection.SelectNext  
(spyAuthenticParagraph).Select();  
// to append a row to a table use AuthenticRange.AppendRow
```

See also

Declaration: `SelectionMoveTabOrder` (*bForward* as Boolean, *bTag* as Boolean)

Description

`SelectionMoveTabOrder()` moves the current selection forwards or backwards.

If *bTag* is false and the current selection is at the last cell of a table a new line will be added.

SelectionSet (obsolete)

Superseded by [AuthenticRange.FirstXMLData](#) and related properties

AuthenticRange supports navigation via XMLData elements as well as navigation by document elements (e.g. characters, words, tags) or text cursor positions.

```
// ----- javascript sample -----  
// instead of:  
// if (! Application.ActiveDocument.DocEditView.SelectionSet(varXMLData1, 0,  
varXMLData2, -1))  
//     MsgBox ("Error: invalid data position");  
// use now:  
try  
{  
    var objSelection = Application.ActiveDocument.AuthenticView.Selection;  
    objSelection.FirstXMLData = varXMLData1;  
    objSelection.FirstXMLdataOffset = 0;  
    objSelection.LastXMLData = varXMLData2;  
    objSelection.LastXMLDataOffset = -1;  
    objSelection.Select();  
}  
catch (err)  
{ MsgBox ("Error: invalid data position"); }  
// to select all text between varXMLData1 and varXMLdata2, inclusive
```

See also

Declaration: `SelectionSet (pStartElement as XMLData, nStartPos as long, pEndElement as XMLData, nEndPos as long) as Boolean`

Description

Use `SelectionSet()` to set a new selection in the Authentic View. Its possible to set `pEndElement` to null (nothing) if the selection should be just over one (`pStartElement`) XML element.

XMLRoot (obsolete)**Superseded by [AuthenticView.XMLDataRoot](#)**

```
// ----- javascript sample -----  
// instead of:  
// var objXMLData = Application.ActiveDocument.DocEditView.XMLRoot;  
// use now:  
var objXMLData = Application.ActiveDocument.AuthenticView.XMLDataRoot;
```

See also**Declaration:** [XMLRoot](#) as [XMLData](#)**Description**

[XMLRoot](#) is the parent element of the currently displayed XML structure. Using the [XMLData](#) interface you have full access to the complete content of the file.

See also [Using XMLData](#) for more information.

3.4 Enumerations

This is a list of all enumerations used by the XMLSpy API. If your scripting environment does not support enumerations use the number-values instead.

3.4.1 ENUMApplicationStatus

Description

Enumeration to specify the current Application status.

Possible values:

eApplicationRunning	= 0
eApplicationAfterLicenseCheck	= 1
eApplicationBeforeLicenseCheck	= 2
eApplicationConcurrentLicenseCheckFailed	= 3
eApplicationProcessingCommandLine	= 4

3.4.2 SPYAttributeTypeDefinition

Description

Attribute type definition that can be selected for generation of Sample XML.

This type is used with the method [GenerateDTDOrSchema](#) and [GenerateDTDOrSchemaEx](#).

Possible values:

spyMergedGlobal	= 0
spyDistinctGlobal	= 1
spyLocal	= 2

3.4.3 SPYAuthenticActions

Description

Actions that can be performed on [AuthenticRange](#) objects.

Possible values:

spyAuthenticInsertAt	= 0
spyAuthenticApply	= 1
spyAuthenticClearSurr	= 2
spyAuthenticAppend	= 3
spyAuthenticInsertBefore	= 4
spyAuthenticRemove	= 5

3.4.4 SPYAuthenticDocumentPosition

Description

Relative and absolute positions used for navigating with [AuthenticRange](#) objects.

Possible values:

spyAuthenticDocumentBegin	= 0
spyAuthenticDocumentEnd	= 1
spyAuthenticRangeBegin	= 2
spyAuthenticRangeEnd	= 3

3.4.5 SPYAuthenticElementActions

Description

Actions that can be used with [GetAllowedElements](#) (superseded by [AuthenticRange.CanPerformActionWith](#)).

Possible values:

k_ActionInsertAt	= 0
k_ActionApply	= 1
k_ActionClearSurr	= 2
k_ActionAppend	= 3
k_ActionInsertBefore	= 4
k_ActionRemove	= 5

3.4.6 SPYAuthenticElementKind

Description

Enumeration of the different kinds of elements used for navigation and selection within the [AuthenticRange](#) and [AuthenticView](#) objects.

Possible values:

spyAuthenticChar	= 0
spyAuthenticWord	= 1
spyAuthenticLine	= 3
spyAuthenticParagraph	= 4
spyAuthenticTag	= 6
spyAuthenticDocument	= 8
spyAuthenticTable	= 9
spyAuthenticTableRow	= 10
spyAuthenticTableColumn	= 11

3.4.7 SPYAuthenticMarkupVisibility

Description

Enumeration values to customize the visibility of markup with [MarkupVisibility](#).

Possible values:

spyAuthenticMarkupHidden	= 0
spyAuthenticMarkupSmall	= 1
spyAuthenticMarkupLarge	= 2
spyAuthenticMarkupMixed	= 3

3.4.8 SPYAuthenticToolbarButtonState

Description

Authentic toolbar button states are given by the following enumeration:

Possible values:

authenticToolbarButtonDefault	= 0
authenticToolbarButtonEnabled	= 1
authenticToolbarButtonDisabled	= 2

3.4.9 SPYDatabaseKind

Description

Values to select different kinds of databases for import. See [DatabaseConnection.DatabaseKind](#) for its use.

Possible values:

spyDB_Access	= 0
spyDB_SQLServer	= 1
spyDB_Oracle	= 2
spyDB_Sybase	= 3
spyDB_MySQL	= 4
spyDB_DB2	= 5
spyDB_Other	= 6
spyDB_Unspecified	= 7
spyDB_PostgreSQL	= 8
spyDB_iSeries	= 9

3.4.10 SPYDialogAction

Description

Values to simulate different interactions on dialogs. See [Dialogs](#) for all dialogs available.

Possible values:

spyDialogOK	= 0	//simulate click on OK button
spyDialogCancel	= 1	//simulate click on Cancel button
spyDialogUserInput	= 2	// show dialog and allow user interaction

3.4.11 SPYDOMType

Description

Enumeration values to parameterize generation of C++ code from schema definitions.

Possible values:

spyDOMType_msxml4	= 0	Obsolete
spyDOMType_xerces	= 1	
spyDOMType_xerces3	= 2	
spyDOMType_msxml6	= 3	

spyDOMType_xerces indicates Xerces 2.x usage; spyDOMType_xerces3 indicates Xerces 3.x usage.

3.4.12 SPYDTDSchemaFormat

Description

Enumeration to identify the different schema formats.

Possible values:

spyDTD	= 0
spyW3C	= 1

3.4.13 SPYEncodingByteOrder

Description

Enumeration values to specify encoding byte ordering for text import and export.

Possible values:

spyNONE	= 0
spyLITTLE_ENDIAN	= 1
spyBIG_ENDIAN	= 2

3.4.14 SPYExportNamespace

Description

Enumeration type to configure handling of namespace identifiers during export.

Possible values:

spyNoNamespace	= 0
spyReplaceColonWithUnderscore	= 1

3.4.15 SPYFindInFilesSearchLocation

Description

The different locations where a search can be performed. This type is used with the [FindInFilesDlg](#) dialog.

Possible values:

spyFindInFiles_Documents	= 0
spyFindInFiles_Project	= 1
spyFindInFiles_Folder	= 2

3.4.16 SPYFrequentElements

Description

Enumeration value to parameterize schema generation.

Possible values:

spyGlobalElements	= 0
spyGlobalComplexType	= 1

3.4.17 SPYImageKind

Description

Enumeration values to parameterize image type of the generated documentation. These values are used in [SchemaDocumentationDiagramFormat](#) and [WSDLDocumentationDiagramFormat](#).

Possible values:

spyImageType_PNG	= 0
spyImageType_EMF	= 1

3.4.18 SPYImportColumnsType

Description

Enumeration to specify different Import columns types.

Possible values:

spyImportColumns_Element	= 0
spyImportColumns_Attribute	= 1

3.4.19 SPYKeyEvent**Description**

Enumeration type to identify the different key events. These events correspond with the equally named windows messages.

Possible values:

spyKeyDown	= 0
spyKeyUp	= 1
spyKeyPressed	= 2

3.4.20 SPYKeyStatus**Description**

Enumeration type to identify the key status.

Possible values:

spyLeftShiftKeyMask	= 1
spyRightShiftKeyMask	= 2
spyLeftCtrlKeyMask	= 4
spyRightCtrlKeyMask	= 8
spyLeftAltKeyMask	= 16
spyRightAltKeyMask	= 32

3.4.21 SPYLibType**Description**

Enumeration values to parameterize generation of C++ code from schema definitions.

Possible values:

spyLibType_static	= 0
spyLibType_dll	= 1

3.4.22 SPYLoading**Description**

Enumeration values to define loading behaviour of URL files.

Possible values:

spyUseCacheProxy	= 0
spyReload	= 1

3.4.23 SPYMouseEvent**Description**

Enumeration type that defines the mouse status during a mouse event. Use the enumeration values as bitmasks rather than directly comparing with them.

Examples

```

' to check for ctrl-leftbutton-down in VB
If (i_eMouseEvent = (XMLSpyLib.spyLeftButtonDownMask Or
XMLSpyLib.spyCtrlKeyDownMask)) Then
' react on ctrl-leftbutton-down
End If

' to check for double-click with any button in VBScript
If (((i_eMouseEvent And spyDoubleClickMask) <> 0) Then
' react on double-click
End If

```

Possible values:

spyNoButtonMask	= 0	
spyMouseMoveMask	= 1	
spyLeftButtonMask	= 2	
spyMiddleButtonMask	= 4	
spyRightButtonMask	= 8	
spyButtonUpMask	= 16	
spyButtonDownMask	= 32	
spyDoubleClickMask	= 64	
spyShiftKeyDownMask	= 128	
spyCtrlKeyDownMask	= 256	
spyLeftButtonDownMask	= 34	// spyLeftButtonMask spyButtonDownMask
spyMiddleButtonDownMask	= 36	// spyMiddleButtonMask spyButtonDownMask
spyRightButtonDownMask	= 40	// spyRightButtonMask spyButtonDownMask
spyLeftButtonUpMask	= 18	// spyLeftButtonMask spyButtonUpMask
spyMiddleButtonUpMask	= 20	// spyMiddleButtonMask spyButtonUpMask
spyRightButtonUpMask	= 24	// spyRightButtonMask spyButtonUpMask
spyLeftDoubleClickMask	= 66	// spyRightButtonMask spyButtonUpMask
spyMiddleDoubleClickMask	= 68	// spyMiddleButtonMask spyDoubleClickMask
spyRightDoubleClickMask	= 72	// spyRightButtonMask spyDoubleClickMask

3.4.24 SPYNumberDateTimeFormat**Description**

Enumeration value to configure database connections.

Possible values:

spySystemLocale	= 0
spySchemaCompatible	= 1

3.4.25 SPYProgrammingLanguage**Description**

Enumeration values to select the programming language for code generation from schema definitions.

Only available/enabled in the Enterprise edition. An error is returned, if accessed by any other version.

Possible values:

spyUndefinedLanguage	= -1
spyJava	= 0
spyCpp	= 1
spyCSharp	= 2

3.4.26 SPYProjectItemTypes

Description

Enumeration values to identify the different elements in project item lists. See [SpyProjectItem.ItemType](#).

Possible values:

spyUnknownItem	= 0
spyFileItem	= 1
spyFolderItem	= 2
spyURLItem	= 3

3.4.27 SPYProjectType

Description

Enumeration values to parameterize generation of C# from schema definitions.

Possible values:

spyVisualStudioProject	= 0	Obsolete
spyVisualStudio2003Project	= 1	Obsolete
spyBorlandProject	= 2	Obsolete
spyMonoMakefile	= 3	
spyVisualStudio2005Project	= 4	For C++ code also
spyVisualStudio2008Project	= 5	For C++ code also
spyVisualStudio2010Project	= 6	For C++ code also

3.4.28 SPYSampleXMLGenerationOptimization

Description

Specify the elements that will be generated in the Sample XML. This enumeration is used in [GenerateSampleXMLDlg](#).

Possible values:

spySampleXMLGen_Optimized	= 0
spySampleXMLGen_NonMandatoryElements	= 1
spySampleXMLGen_Everything	= 2

3.4.29 SPYSampleXMLGenerationSchemaOrDTDAssignment

Description

Specifies what kind of reference to the schema/DTD should be added to the generated Sample XML.

This enumeration is used in [GenerateSampleXMLDlg](#).

Possible values:

spySampleXMLGen_AssignRelatively	= 0
spySampleXMLGen_AssignAbsolutely	= 1
spySampleXMLGen_DoNotAssign	= 2

3.4.30 SPYSchemaDefKind

Description

Enumeration type to select schema diagram types.

Possible values:

spyKindElement	= 0
spyKindComplexType	= 1
spyKindSimpleType	= 2
spyKindGroup	= 3
spyKindModel	= 4
spyKindAny	= 5
spyKindAttr	= 6
spyKindAttrGroup	= 7
spyKindAttrAny	= 8
spyKindIdentityUnique	= 9
spyKindIdentityKey	= 10
spyKindIdentityKeyRef	= 11
spyKindIdentitySelector	= 12
spyKindIdentityField	= 13
spyKindNotation	= 14
spyKindInclude	= 15
spyKindImport	= 16
spyKindRedefine	= 17
spyKindFacet	= 18
spyKindSchema	= 19
spyKindCount	= 20

3.4.31 SPYSchemaDocumentationFormat**Description**

Enumeration values to parameterize generation of schema documentation. These values are used in [SchemaDocumentationDebugOutputFormat](#) and [WSDLDocumentationDebugOutputFormat](#).

Possible values:

spySchemaDoc_HTML	= 0
spySchemaDoc_MSWord	= 1
spySchemaDoc_RTF	= 2
spySchemaDoc_PDF	= 3

3.4.32 SPYSchemaExtensionType**Description**

Enumeration to specify different Schema Extension types.

Possible values:

spySchemaExtension_None	= 0
spySchemaExtension_SQL_XML	= 1
spySchemaExtension_MS_SQL_Server	= 2
spySchemaExtension_Oracle	= 3

3.4.33 SPYSchemaFormat**Description**

Enumeration to specify different Schema Format types.

Possible values:

spySchemaFormat_Hierarchical	= 0

spySchemaFormat_Flat = 1

3.4.34 SPYTextDelimiters

Description

Enumeration values to specify text delimiters for text export.

Possible values:

spyTabulator	= 0
spySemicolon	= 1
spyComma	= 2
spySpace	= 3

3.4.35 SPYTextEnclosing

Description

Enumeration value to specify text enclosing characters for text import and export.

Possible values:

spyNoEnclosing	= 0
spySingleQuote	= 1
spyDoubleQuote	= 2

3.4.36 SPYTypeDetection

Description

Enumeration to select how type detection works during [GenerateDTDOrSchema](#) and [GenerateDTDOrSchemaEx](#).

Possible values:

spyBestPossible	= 0
spyNumbersOnly	= 1
spyNoDetection	= 2

3.4.37 SPYURLTypes

Description

Enumeration to specify different URL types.

Possible values:

spyURLTypeAuto	= -1
spyURLTypeXML	= 0
spyURLTypeDTD	= 1

3.4.38 SPYViewModes

Description

Enumeration values that define the different view modes for XML documents. The mode *spyViewAuthentic(4)* identifies the mode that was intermediately called DocEdit mode and is now called Authentic mode. The mode *spyViewWSDL* identifies a mode which is mapped to the schema view on the GUI but distinguished internally.

Possible values:

spyViewGrid	= 0
spyViewText	= 1
spyViewBrowser	= 2
spyViewSchema	= 3

spyViewContent	= 4	// obsolete
spyViewAuthentic	= 4	
spyViewWSDL	= 5	
spyViewZIP	= 6	
spyViewEditionInfo	= 7	
spyViewXBRL	= 8	

3.4.39 SPYVirtualKeyMask

Description

Enumeration type for the most frequently used key masks that identify the status of the virtual keys. Use these values as bitmasks rather than directly comparing with them. When necessary, you can create further masks by using the 'logical or' operator.

Examples

```
' VBScript sample: check if ctrl-key is pressed
If ((i_nVirtualKeyStatus And spyCtrlKeyMask) <> 0)) Then
    ' ctrl-key is pressed
End If

' VBScript sample: check if ONLY ctrl-key is pressed
If (i_nVirtualKeyStatus == spyCtrlKeyMask) Then
    ' exactly ctrl-key is pressed
End If

// JScript sample: check if any of the right virtual keys is pressed
if ((i_nVirtualKeyStatus & (spyRightShiftKeyMask | spyRightCtrlKeyMask |
spyRightAltKeyMask)) != 0)
{
    ; ' right virtual key is pressed
}
```

Possible values:

spyNoVirtualKeyMask	= 0	
spyLeftShiftKeyMask	= 1	
spyRightShiftKeyMask	= 2	
spyLeftCtrlKeyMask	= 4	
spyRightCtrlKeyMask	= 8	
spyLeftAltKeyMask	= 16	
spyRightAltKeyMask	= 32	
spyShiftKeyMask	= 3	// spyLeftShiftKeyMask spyRightShiftKeyMask
spyCtrlKeyMask	= 12	// spyLeftCtrlKeyMask spyRightCtrlKeyMask
spyAltKeyMask	= 48	// spyLeftAltKeyMask spyRightAltKeyMask

3.4.40 SPYXMLDataKind

Description

The different types of XMLData elements available for XML documents.

Possible values:

spyXMLDataXMLDocStruct	= 0
spyXMLDataXMLEntityDocStruct	= 1
spyXMLDataDTDDocStruct	= 2
spyXMLDataXML	= 3
spyXMLDataElement	= 4
spyXMLDataAttr	= 5
spyXMLDataText	= 6

spyXMLDataCData	= 7
spyXMLDataComment	= 8
spyXMLDataPI	= 9
spyXMLDataDefDoctype	= 10
spyXMLDataDefExternalID	= 11
spyXMLDataDefElement	= 12
spyXMLDataDefAttlist	= 13
spyXMLDataDefEntity	= 14
spyXMLDataDefNotation	= 15
spyXMLDataKindsCount	= 16

3.5 Application API for Java

The objects described in this section (Application API for Java) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: [Programming Languages | Java](#).

The Application API in Java has an interface built up of Java classes, each of which corresponds to an object in the [Application API](#). Developers can use these Java classes to interact with the COM API. These classes are listed below and described in subsequent sections. For a description of the [Application API](#) objects themselves, see the [Application API documentation](#). Bear in mind that some API features are only available in scripting environments; these have therefore not been ported to Java.

Java classes

- [SpyApplication](#)
- [SpyProject](#)
- [SpyProjectItems](#)
- [SpyProjectItem](#)
- [SpyDocuments](#)
- [SpyDoc](#)
- [SpyAuthenticView](#)
- [SpyAuthenticRange](#)
- [SpyDocEditView](#)
- [SpyDocEditSelection](#)
- [SpyGridView](#)
- [SpyTextView](#)
- [SpyXMLData](#)
- [SpyDialogs](#)
- [SpyCodeGeneratorDlg](#)
- [SpyDTDSchemaGeneratorDlg](#)
- [SpyFileSelectionDlg](#)
- [SpyFindInFilesDlg](#)
- [SpyGenerateSampleXMLDlg](#)
- [SpySchemaDocumentationDlg](#)
- [SpyWSDL20DocumentationDlg](#)
- [SpyWSDLDocumentationDlg](#)
- [SpyXBRLDocumentationDlg](#)
- [SpyDatabaseConnection](#)
- [SpyElementList](#)
- [SpyElementListItem](#)
- [SpyExportSettings](#)
- [SpyFindInFilesResults](#)
- [SpyFindInFilesResult](#)
- [SpyFindInFilesMatch](#)
- [SpyTextImportExportSettings](#)

Implementation of COM properties in Java

Properties in Java have been defined to include both a **set** and **get** method (set if it is allowed by the COM implementation). For example, the COM class `Document` contains the `GridView`

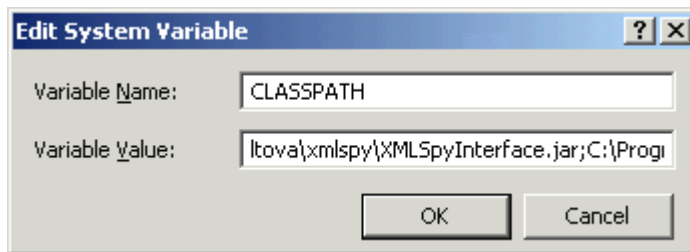
property. In Java the method is called `spyDoc` and the property is defined as a `GetGridView` method.

If you encounter compiling problems, please check the following points:

- The `xmlspylib.dll` must be available in `.. \windows\system32`.
- The `XMLSpyInterface.jar` file must be inserted in the `ClassPath` environment variable.

Setting the `ClassPath` variable in Windows XP

1. Click **Start | Settings | Control panel | System | Advanced | Environment Variables**. This opens the Environment Variables dialog box.
2. If a `ClassPath` entry already exists in the **System variables** group, select the `ClassPath` entry, and click the **Edit** button. Edit the path to: `"C: \Program Files\Altova\xmlspy\XMLSpyInterface.jar"`.



If a `ClassPath` entry does not exist in the System variables group, click the **New** button. The New System Variable dialog pops up. Enter `CLASSPATH` as the variable name, and `"C: \Program Files\Altova\xmlspy\XMLSpyInterface.jar"` as the `ClassPath` variable (alter the path to match your installation, if necessary).

3.5.1 Sample source code

The `"SpyDoc doc = app.GetDocuments().OpenFile(...)"` command parameter must be altered to suit your environment.

What the sample does:

- Starts a new XMLSpy instance
- Opens the Datasheet.xml file (alter the path here...)
- Switches to the Enhanced Grid view
- Appends a new child element called "NewChild" with the text value "NewValuE" element to the root element
- Checks if the document is valid and outputs a message to the Java console
- Quits and releases the XMLSpy application

```
import XMLSpyInterface.*;

public class TestSpyInterface
{
    public TestSpyInterface() {}

    public static void main(String[] args)
    {
        SpyApplication app = null;
        SpyDoc oDoc = null;
```

```

SpyXMLData oData = null;
SpyXMLData oNewChild = null;

try
{
    app = new SpyApplication();
    app.ShowApplication( true );

    oDoc = app.GetDocuments().OpenFile( "C:\\FilePath\\OrgChart.xml",
    true );

    // OrgChart.xml is in the folder C:\Documents and
    Settings\<username>\My Documents\Altova\XMLSpy2012\. The filepath
    should be in
    // the form: C:\\Documents and
    Settings\\Username\\Folder\\Filename.xml

    if ( oDoc != null )
    {
        oDoc.SwitchViewMode( SPYViewModes.spyViewGrid);
        oData = oDoc.GetRootElement();
        oNewChild = oDoc.CreateChild( SPYXMLDataKind.spyXMLDataElement);

        oNewChild.SetName( "NewChild" );
        oNewChild.SetTextValue( "newValuE");
        oData.AppendChild( oNewChild);

        if ( oDoc.IsValid() == false )
        {
            // is to be expected after above insertion
            System.out.println( "!!!!!!validation error: " +
            oDoc.GetErrorString() );
            System.out.println( "!!!!!!validation error: " +
            oDoc.GetErrorPos() );
            System.out.println( "!!!!!!validation error: " +
            oDoc.GetBadData() );
        }
    }

    app.Quit();
}
finally
{
    // Free any allocated resources by calling ReleaseInstance().
    if ( oNewChild != null )
        oNewChild.ReleaseInstance();

    if ( oData != null )
        oData.ReleaseInstance();

    if ( oDoc != null )
        oDoc.ReleaseInstance();

    if ( app != null )
        app.ReleaseInstance();
}
}

```

If you have difficulties compiling this sample, please try the following commands on the (**Start | Run | cmd**) command line. Please make sure you are currently in the folder that contains the sample java file.

compilation

```
javac -classpath c:\yourpathhere\XMLSpyInterface.jar testspyinterface.java
```

Execution

```
java -classpath c:\yourpathhere\XMLSpyInterface.jar testspyinterface
```

3.5.2 SpyApplication

```
public class SpyApplication
{
    public void ReleaseInstance();
    public void ShowApplication( boolean bShow );
    public void Quit();
    public void AddMacroMenuItem( String sMacro, String sDisplayText );
    public void ClearMacroMenu();
    public SpyDoc GetActiveDocument();
    public SpyProject GetCurrentProject();
    public SpyDocuments GetDocuments();
    public SpyElementList GetDatabaseImportElementList( SpyDatabaseConnection
oImportSettings );
    public SpyDatabaseConnection GetDatabaseSettings();
    public SpyElementList GetDatabaseTables( SpyDatabaseConnection
oImportSettings );
    public SpyExportSettings GetExportSettings();
    public SpyElementList GetTextImportElementList( SpyTextImportExportSettings
oImportSettings );
    public SpyTextImportExportSettings GetTextImportExportSettings();
    public SpyDoc ImportFromDatabase( SpyDatabaseConnection oImportSettings,
SpyElementList oElementList );
    public SpyDoc ImportFromSchema( SpyDatabaseConnection oImportSettings,
String strTable, SpyDoc oSchemaDoc );
    public SpyDoc ImportFromText( SpyTextImportExportSettings oImportSettings,
SpyElementList oElementList );
    public SpyDoc ImportFromWord( String sFile );
    public void NewProject( String sPath, boolean bDiscardCurrent );
    public void OpenProject( String sPath, boolean bDiscardCurrent, boolean
bDialog );
    public long ShowForm( String sName );
    public void URLDelete( String sURL, String sUser, String sPassword );
    public void URLMakeDirectory( String sURL, String sUser, String sPassword );

    public int GetWarningNumber();
    public String GetWarningText();

    // since Version 2004R4
    public SpyApplication GetApplication();
    public SpyApplication GetParent();
    public SpyDialogs GetDialogs();
    public boolean GetVisible();
    public void SetVisible( boolean i_bVisibility );
    public long GetWindowHandle();

    public void ReloadSettings();
    public SpyFindInFilesResults FindInFiles( SpyFindInFilesDlg dlgSettings );
    public boolean ShowFindInFiles( SpyFindInFilesDlg dlgSettings );
    public void Selection( String sVal );

    public long Status();
    public int MajorVersion();
    public int MinorVersion();
    public String Edition();
    public boolean IsAPISupported();
}
```

```

    public long ServicePackVersion();
    public void CreateXMLSchemaFromDBStructure( SpyDatabaseConnection
oConnection, SpyElementList oTables );
}

```

3.5.3 SpyCodeGeneratorDlg

Only available/enabled in the Enterprise edition. An error is returned, if accessed by any other version.

```

// since version 2004R4
public class SpyCodeGeneratorDlg
{
    public void ReleaseInstance();
    public SpyApplication GetApplication();
    public SpyDialogs GetParent();
    public long GetProgrammingLanguage();
    public void SetProgrammingLanguage( long i_eVal );
    public String GetTemplateFileName();
    public void SetTemplateFileName( String i_strVal );
    public String GetOutputPath();
    public void SetOutputPath( String i_strVal );
    public long GetOutputPathDialogAction();
    public void SetOutputPathDialogAction( long i_eVal );
    public long GetPropertySheetDialogAction();
    public void SetPropertySheetDialogAction( long i_eVal );
    public long GetOutputResultDialogAction();
    public void SetOutputResultDialogAction( long i_eVal );
    public long GetCPPSettings\_DOMType();
    public void SetCPPSettings\_DOMType( long i_eVal );
    public long GetCPPSettings\_LibraryType();
    public void SetCPPSettings\_LibraryType( long i_eVal );
    public boolean GetCPPSettings\_UseMFC();
    public void SetCPPSettings\_UseMFC( boolean i_bVal );
    public long GetCSharpSettings\_ProjectType();
    public void SetCSharpSettings\_ProjectType( long i_eVal );
}

```

3.5.4 SpyDatabaseConnection

```

public class SpyDatabaseConnection
{
    public void ReleaseInstance();
    public String GetADOConnection();
    public void SetADOConnection( String sValue );
    public boolean GetAsAttributes();
    public void SetAsAttributes( boolean bValue );
    public boolean GetCreateMissingTables();
    public void SetCreateMissingTables( boolean bValue );
    public boolean GetCreateNew();
    public void SetCreateNew( boolean bValue );
    public boolean GetExcludeKeys();
    public void SetExcludeKeys( boolean bValue );
    public String GetFile();
    public void SetFile( String sValue );
    public boolean GetIncludeEmptyElements();
    public void SetIncludeEmptyElements( boolean bValue );
    public long GetNumberDateTimeFormat();
    public void SetNumberDateTimeFormat( long nValue );
}

```

```

public String GetODBCConnection();
public void SetODBCConnection( String sValue );
public String GetSQLSelect();
public void SetSQLSelect( String sValue );
public long GetTextFieldLen();
public void SetTextFieldLen( long nValue );

// since version 2004R4
public long GetDatabaseKind();
public void SetDatabaseKind( long nValue );

// since version 2008R2
public boolean GetCommentIncluded();
public void SetCommentIncluded( boolean bValue );
public String GetNullReplacement();
public void SetNullReplacement( String sValue );
public String GetDatabaseSchema();
public void SetDatabaseSchema( String sValue );

// since version 2010r3
public boolean GetPrimaryKeys()
public void SetPrimaryKeys( boolean bValue )
public boolean GetForeignKeys()
public void SetForeignKeys( boolean bValue )
public boolean GetUniqueKeys()
public void SetUniqueKeys( boolean bValue )
public long GetSchemaExtensionType()
public void SetSchemaExtensionType( long nValue )
public long GetSchemaFormat()
public void SetSchemaFormat( long nValue )
public long GetImportColumnsType()
public void SetImportColumnsType( long nValue )
}

```

3.5.5 SpyDialogs

```

// Since version 2004R4
public class SpyDialogs
{
    public SpyApplication GetApplication();
    public SpyApplication GetParent();
    public SpyCodeGeneratorDlg GetCodeGeneratorDlg();
    public SpyFileSelectionDlg GetFileSelectionDlg();
    public SpySchemaDocumentationDlg GetSchemaDocumentationDlg();
    public SpyGenerateSampleXMLDlg GetGenerateSampleXMLDlg();
    public SpyDTDSchemaGeneratorDlg GetDTDSchemaGeneratorDlg();
    public SpyFindInFilesDlg GetFindInFilesDlg();
    public SpyWSDLDocumentationDlg GetWSDLDocumentationDlg();

    // Since version 2010
    public SpyWSDL20DocumentationDlg GetWSDL20DocumentationDlg();
    public SpyXBRLDocumentationDlg GetXBRLDocumentationDlg();
}

```

3.5.6 SpyDoc

```

public class SpyDoc
{
    public void ReleaseInstance();
}

```

```

public void SetEncoding( String strEncoding );
public void SetPathName( String strPath );
public String GetPathName();
public String GetTitle();
public boolean IsModified();
public void Save();
public void Close( boolean bDiscardChanges );
public void UpdateViews();
public long GetCurrentViewMode();
public boolean SwitchViewMode( long nMode );
public SpyGridView GetGridView();
public void SetActiveDocument();
public void StartChanges();
public void EndChanges();
public void TransformXSL();
public void AssignDTD( String sDTDFile, boolean bDialog );
public void AssignSchema( String sSchemaFile, boolean bDialog );
public void AssignXSL( String sXSLFile, boolean bDialog );
public void ConvertDTDOrSchema( long nFormat, long nFrequentElements );
public SpyXMLData CreateChild( long nKind );
public void CreateSchemaDiagram( long nKind, String sName, String sFile );
public SpyDocEditView GetDocEditView();
public void ExportToDatabase( SpyXMLData oFromChild, SpyExportSettings
oExportSettings, SpyDatabaseConnection oDatabaseConnection );
public void ExportToText( SpyXMLData oFromChild, SpyExportSettings
oExportSettings, SpyTextImportExportSettings oTextSettings );
public void GenerateDTDOrSchema( long nFormat, int nValuesList, long
nDetection, long nFrequentElements );
public SpyElementList GetExportElementList( SpyXMLData oFromChild,
SpyExportSettings oExportSettings );
public SpyXMLData GetRootElement();
public String SaveInString( SpyXMLData oData, boolean bMarked );
public void SaveToURL( String sUrl, String sUser, String sPassword );
public String GetErrorString(); // See IsValid() or IsWellFormed()
public int GetErrorPos(); // See IsValid() or IsWellFormed()
public SpyXMLData GetBadData(); // See IsValid() or IsWellFormed()
public boolean IsValid();
public boolean IsWellFormed( SpyXMLData oData, boolean bWithChildren );

// Since version 2004R3
public SpyAuthenticView GetAuthenticView()

// Since version 2004R4
public SpyApplication GetApplication();
public SpyDocuments GetParent();
public String GetFullName();
public void SetFullName( String i_strName );
public String GetName();
public String GetPath();
public boolean GetSaved();
public void SaveAs( String i_strFileNameOrPath );
public String GetEncoding();
public SpyXMLData GetDataRoot();
public void GenerateProgramCode( SpyCodeGeneratorDlg i_dlg );
public void AssignXSLFO( String i_strFile, boolean i_bUseDialog );
public void TransformXSLFO();
public void GenerateSchemaDocumentation( SpySchemaDocumentationDlg i_dlg );

public void ExecuteXQuery( String i_strXMLSourceFile );
public void SetExternalIsValid( boolean bIsValid );
public SpyDoc GenerateSampleXML( SpyGenerateSampleXMLDlg ipGenerateXMLDlg );
public boolean UpdateXMLData();
public String GetAsXMLString();

```

```

public void SetAsXMLString( String newVal );
public SpyDoc GenerateDTDOrSchemaEx( SpyDTDSchemaGeneratorDlg
ipDTDSchemaGeneratorDlg );
public SpyDoc ConvertDTDOrSchemaEx( long nFormat, long nFrequentElements,
String sOutputPath, long nOutputPathDialogAction );
public SpyTextView GetTextView();
public String[] GetSuggestions();
public void SetSuggestions( String[] aList );
public void SetSelection( String sVal );

// Since version 2009
public void GenerateWSDLDocumentation( SpyWSDLDocumentationDlg
ipWSDLDocumenationDlg );
public void TransformXSLEx( long nDialogAction );

// Since version 2010
public void GenerateWSDL20Documentation( SpyWSDL20DocumentationDlg
ipWSD20DocumenationDlg );
public void GenerateXBRLDocumentation( SpyXBRLDocumentationDlg
ipXBRLDocumentationDlg );
public SpyDoc ConvertToWSDL20( String sFilePath, boolean bShowDialogs );

// Since version 2010r3
public String CreateDBStructureFromXMLSchema( SpyDatabaseConnection
oConnection, SpyElementList oTables, boolean bDropTableWithExistingName );
public SpyElementList GetDBStructureList( SpyDatabaseConnection oConnection
);
}

```

3.5.7 SpyDocuments

```

public class SpyDocuments
{
    public void ReleaseInstance();
    public long Count();
    public SpyDoc GetItem( long nNo );
    public SpyDoc NewFile( String strFile, String strType );
    public SpyDoc NewFileFromText( String nSource, String strType );
    public SpyDoc OpenFile( String sPath, boolean bDialog );
    public SpyDoc OpenURL( String sUrl, long nURLType, long nLoading, String
sUser, String sPassword );
    public SpyDoc OpenURLDialog(String sURL, long nURLType, long nLoading,
String sUser, String sPassword );
    // Since version 2011r2
    public SpyDoc NewAuthenticFile( String strSPSPath, String strXMLPath );
    public SpyDoc OpenAuthenticFile( String strSPSPath, String strXMLPath );
}

```

3.5.8 SpyDTDSchemaGeneratorDlg

```

public class SpyDTDSchemaGeneratorDlg
{
    public void ReleaseInstance();
    public SpyApplication GetApplication();
    public long GetDTDSchemaFormat();
    public void SetDTDSchemaFormat( long newVal );
    public short GetValueList();
    public void SetValueList( short newVal );
    public long GetTypeDetection();
    public void SetTypeDetection( long newVal );
    public long GetFrequentElements();
}

```



```

    public void SetFrequentElements( long newVal );
    public boolean GetMergeAllEqualNamed();
    public void SetMergeAllEqualNamed( boolean newVal );
    public boolean GetResolveEntities();
    public void SetResolveEntities( boolean newVal );
    public long GetAttributeTypeDefinition();
    public void SetAttributeTypeDefinition( long newVal );
    public boolean GetGlobalAttributes();
    public void SetGlobalAttributes( boolean newVal );
    public boolean GetOnlyStringEnums();
    public void SetOnlyStringEnums( boolean newVal );
    public long GetMaxEnumLength();
    public void SetMaxEnumLength( long newVal );
    public String GetOutputPath();
    public void SetOutputPath( String newVal );
    public long GetOutputPathDialogAction();
    public void SetOutputPathDialogAction( long newVal );
}

```

3.5.9 SpyElementList

```

public class SpyElementList
{
    public void ReleaseInstance();
    public long GetCount();
    public SpyElementListItem GetItem( long nIndex );
    public void RemoveElement( long nIndex );
}

```

3.5.10 SpyElementListItem

```

public class SpyElementListItem
{
    public void ReleaseInstance();
    public long GetElementKind();
    public void SetElementKind( long nKind );
    public long GetFieldCount();
    public String GetName();
    public long GetRecordCount();
}

```

3.5.11 SpyExportSettings

```

public class SpyExportSettings
{
    public void ReleaseInstance();
    public boolean GetCreateKeys();
    public void SetCreateKeys( boolean bValue );
    public SpyElementList GetElementList();
    public void SetElementList( SpyElementList obj );
    public boolean GetEntitiesToText();
    public void SetEntitiesToText( boolean bValue );
    public boolean GetExportAllElements();
    public void SetExportAllElements( boolean bValue );
    public boolean GetFromAttributes();
    public void SetFromAttributes( boolean bValue );
    public boolean GetFromSingleSubElements();
    public void SetFromSingleSubElements( boolean bValue );
    public boolean GetFromTextValues();
}

```

```

    public void SetFromTextValues( boolean bValue );
    public boolean GetIndependentPrimaryKey();
    public void SetIndependentPrimaryKey( boolean bValue );
    public long GetNamespace();
    public void SetNamespace( long nValue );
    public int GetSubLevelLimit();
    public void SetSubLevelLimit( int nValue );
}

```

3.5.12 SpyFileSelectionDlg

```

// Since version 2004R4
public class SpyFileSelectionDlg
{
    public void ReleaseInstance();
    public SpyApplication GetApplication();
    public SpyDialogs GetParent();
    public String GetFullName();
    public void SetFullName( String i_strName );
    public long GetDialogAction();
    public void SetDialogAction( long i_eAction );
}

```

3.5.13 SpyFindInFilesDlg

```

public class SpyFindInFilesDlg
{
    public void ReleaseInstance();
    public SpyApplication GetApplication();
    public String GetFind();
    public void SetFind( String sNewVal );
    public boolean GetRegularExpression();
    public void SetRegularExpression( boolean bNewVal );
    public String GetReplace();
    public void SetReplace( String sNewVal );
    public boolean GetReplaceOnDisk();
    public void SetReplaceOnDisk( boolean bNewVal );
    public boolean GetDoReplace();
    public void SetDoReplace( boolean bNewVal );
    public boolean GetMatchWholeWord();
    public void SetMatchWholeWord( boolean bNewVal );
    public boolean GetMatchCase();
    public void SetMatchCase( boolean bNewVal );
    public long GetSearchLocation();
    public void SetSearchLocation( long nPosition );
    public String GetStartFolder();
    public void SetStartFolder( String sNewVal );
    public boolean GetIncludeSubfolders();
    public void SetIncludeSubfolders( boolean bNewVal );
    public boolean GetSearchInProjectFilesDoExternal();
    public void SetSearchInProjectFilesDoExternal( boolean bNewVal );
    public String GetFileExtension();
    public void SetFileExtension( String sNewVal );
    public boolean GetAdvancedXMLSearch();
    public void SetAdvancedXMLSearch( boolean bNewVal );
    public boolean GetXMLElementNames();
    public void SetXMLElementNames( boolean bNewVal );
    public boolean GetXMLElementContents();
    public void SetXMLElementContents( boolean bNewVal );
    public boolean GetXMLAttributeNames();
}

```

```

    public void SetXMLAttributeNames( boolean bNewVal );
    public boolean GetXMLAttributeContents();
    public void SetXMLAttributeContents( boolean bNewVal );
    public boolean GetXMLComments();
    public void SetXMLComments( boolean bNewVal );
    public boolean GetXMLCDATA();
    public void SetXMLCDATA( boolean bNewVal );
    public boolean GetXMLPI();
    public void SetXMLPI( boolean bNewVal );
    public boolean GetXMLRest();
    public void SetXMLRest( boolean bNewVal );
    public boolean GetShowResult();
    public void SetShowResult( boolean bNewVal );
}

```

3.5.14 SpyFindInFilesMatch

```

public class SpyFindInFilesMatch
{
    public void ReleaseInstance();
    public long Line();
    public long Position();
    public long Length();
    public String LineText();
    public boolean Replaced();
}

```

3.5.15 SpyFindInFilesResult

```

public class SpyFindInFilesResult
{
    public void ReleaseInstance();
    public long Count();
    public SpyFindInFilesMatch GetItem( long nNo );
    public String GetPath();
    public SpyDoc GetDocument();
}

```

3.5.16 SpyFindInFilesResults

```

public class SpyFindInFilesResults
{
    public void ReleaseInstance();
    public long Count();
    public SpyFindInFilesResult GetItem( long nNo );
}

```

3.5.17 SpyGenerateSampleXMLDlg

```

public class SpyGenerateSampleXMLDlg
{
    public void ReleaseInstance();
    public SpyApplication GetApplication();
    public boolean GetNonMandatoryAttributes();
    public void SetNonMandatoryAttributes( boolean newVal );
    public boolean GetNonMandatoryElements();
    public void SetNonMandatoryElements( boolean newVal );
    public boolean GetTakeFirstChoice();
}

```

```

    public void SetTakeFirstChoice( boolean newVal );
    public long GetRepeatCount();
    public void SetRepeatCount( long newVal );
    public boolean GetFillWithSampleData();
    public void SetFillWithSampleData( boolean newVal );
    public boolean GetFillElementsWithSampleData();
    public void SetFillElementsWithSampleData( boolean newVal );
    public boolean GetFillAttributesWithSampleData();
    public void SetFillAttributesWithSampleData( boolean newVal );
    public boolean GetContentOfNillableElementsIsNonMandatory();
    public void SetContentOfNillableElementsIsNonMandatory( boolean newVal );
    public boolean GetTryToUseNonAbstractTypes();
    public void SetTryToUseNonAbstractTypes( boolean newVal );
    public long GetOptimization();
    public void SetOptimization( long newVal );
    public long GetSchemaOrDTDAssignment();
    public void SetSchemaOrDTDAssignment( long newVal );
    public String GetLocalNameOfRootElement();
    public void SetLocalNameOfRootElement( String newVal );
    public String GetNamespaceURIOfRootElement();
    public void SetNamespaceURIOfRootElement( String newVal );
    public long GetOptionsDialogAction();
    public void SetOptionsDialogAction( long newVal );
}

```

3.5.18 SpyGridView

```

public class SpyGridView
{
    public void ReleaseInstance();
    public SpyXMLData GetCurrentFocus();
    public void Deselect( SpyXMLData oData );
    public boolean GetIsVisible();
    public void Select( SpyXMLData oData );
    public void SetFocus( SpyXMLData oData );
}

```

3.5.19 SpyProject

```

public class SpyProject
{
    public void ReleaseInstance();
    public void CloseProject( boolean bDiscardChanges, boolean bCloseFiles,
boolean bDialog );
    public String GetProjectFile();
    public void SetProjectFile( String sFile );
    public SpyProjectItems GetRootItems();
    public void SaveProject();
    public void SaveProjectAs( String sPath, boolean bDialog );
}

```

3.5.20 SpyProjectItem

```

public class SpyProjectItem
{
    public void ReleaseInstance();
    public SpyProjectItems GetChildItems();
    public String GetFileExtensions();
    public void SetFileExtensions( String sExtensions );
}

```

```

    public long GetItemType\(\);
    public String GetName\(\);
    public SpyDoc Open\(\);
    public SpyProjectItem GetParentItem\(\);
    public String GetPath\(\);
    public String GetValidateWith\(\);
    public void SetValidateWith( String sVal );
    public String GetXMLForXSLTransformation\(\);
    public void SetXMLForXSLTransformation( String sVal );
    public String GetXSLForXMLTransformation\(\);
    public void SetXSLForXMLTransformation( String sVal );
    public String GetXSLTransformationFileExtension\(\);
    public void SetXSLTransformationFileExtension( String sVal );
    public String GetXSLTransformationFolder\(\);
    public void SetXSLTransformationFolder( String sVal );
}

```

3.5.21 SpyProjectItems

```

public class SpyProjectItems
{
    public void ReleaseInstance\(\);
    public void AddFile( String sPath );
    public void AddFolder( String sName );
    public void AddURL( String sURL, long nURLType, String sUser, String
sPassword, boolean bSave );
    public long Count\(\);
    public SpyProjectItem GetItem( long nNumber );
    public void RemoveItem( SpyProjectItem oItemToRemove );
}

```

3.5.22 SpySchemaDocumentationDlg

```

// Since version 2004R4
public class SpySchemaDocumentationDlg
{
    public void ReleaseInstance\(\);
    public SpyApplication GetApplication\(\);
    public SpyDialogs GetParent\(\);

    public String GetOutputFile\(\);
    public void SetOutputFile( String i_strVal );
    public long GetOutputFormat\(\);
    public void SetOutputFormat( long i_eVal );

    public boolean GetShowResult\(\);
    public void SetShowResult( boolean i_bVal );
    public long GetOptionsDialogAction\(\);
    public void SetOptionsDialogAction( long i_eVal );
    public long GetOutputFileDialogAction\(\);
    public void SetOutputFileDialogAction( long i_eVal );
    public boolean GetShowProgressBar\(\);
    public void SetShowProgressBar( boolean i_bVal );

    public void IncludeAll( boolean i_bInclude );
    public boolean GetIncludeIndex\(\);
    public void SetIncludeIndex( boolean i_bVal );
    public boolean GetIncludeGlobalElements\(\);
    public void SetIncludeGlobalElements( boolean i_bVal );
    public boolean GetIncludeLocalElements\(\);
}

```

```
public void SetIncludeLocalElements( boolean i_bVal );
public boolean GetIncludeGroups();
public void SetIncludeGroups( boolean i_bVal );
public boolean GetIncludeComplexTypes();
public void SetIncludeComplexTypes( boolean i_bVal );
public boolean GetIncludeSimpleTypes();
public void SetIncludeSimpleTypes( boolean i_bVal );
public boolean GetIncludeAttributeGroups();
public void SetIncludeAttributeGroups( boolean i_bVal );
public boolean GetIncludeRedefines();
public void SetIncludeRedefines( boolean i_bVal );

public void AllDetails( boolean i_bDetailsOn );
public boolean GetShowDiagram();
public void SetShowDiagram( boolean i_bVal );
public boolean GetShowNamespace();
public void SetShowNamespace( boolean i_bVal );
public boolean GetShowType();
public void SetShowType( boolean i_bVal );
public boolean GetShowChildren();
public void SetShowChildren( boolean i_bVal );
public boolean GetShowUsedBy();
public void SetShowUsedBy( boolean i_bVal );
public boolean GetShowProperties();
public void SetShowProperties( boolean i_bVal );
public boolean GetShowSingleFacets();
public void SetShowSingleFacets( boolean i_bVal );
public boolean GetShowPatterns();
public void SetShowPatterns( boolean i_bVal );
public boolean GetShowEnumerations();
public void SetShowEnumerations( boolean i_bVal );
public boolean GetShowAttributes();
public void SetShowAttributes( boolean i_bVal );
public boolean GetShowIdentityConstraints();
public void SetShowIdentityConstraints( boolean i_bVal );
public boolean GetShowAnnotations();
public void SetShowAnnotations( boolean i_bVal );
public boolean GetShowSourceCode();
public void SetShowSourceCode( boolean i_bVal );

// Since version 2009
public boolean GetEmbedDiagrams();
public void SetEmbedDiagrams( boolean i_bVal );
public long GetDiagramFormat();
public void SetDiagramFormat( long i_nVal );
public boolean GetIncludeGlobalAttributes();
public void SetIncludeGlobalAttributes( boolean i_bVal );
public boolean GetIncludeLocalAttributes();
public void SetIncludeLocalAttributes( boolean i_bVal );
public boolean GetIncludeReferencedSchemas();
public void SetIncludeReferencedSchemas( boolean i_bVal );
public boolean GetMultipleOutputFiles();
public void SetMultipleOutputFiles( boolean i_bVal );

// Since version 2010
public boolean GetEmbedCSSInHTML();
public void SetEmbedCSSInHTML( boolean i_bVal );
public boolean GetCreateDiagramsFolder();
public void SetCreateDiagramsFolder( boolean i_bVal );

// Since version 2010r3
public boolean GetGenerateRelativeLinks();
public void SetGenerateRelativeLinks( boolean i_bVal );
```

```

// Since version 2011r2
public boolean GetUseFixedDesign();
public void SetUseFixedDesign( boolean i_bVal );
public String GetSPSFile();
public void SetSPSFile( String i_strVal );
}

```

3.5.23 SpyTextImportExportSettings

```

public class SpyTextImportExportSettings
{
    public void ReleaseInstance();
    public String GetDestinationFolder();
    public void SetDestinationFolder( String sVal );
    public long GetEnclosingCharacter();
    public void SetEnclosingCharacter( long nEnclosing );
    public String GetEncoding();
    public void SetEncoding( String sVal );
    public long GetEncodingByteOrder();
    public void SetEncodingByteOrder( long nByteOrder );
    public long GetFieldDelimiter();
    public void SetFieldDelimiter( long nDelimiter );
    public String GetFileExtension();
    public void SetFileExtension( String sVal );
    public boolean GetHeaderRow();
    public void SetHeaderRow( boolean bVal );
    public String GetImportFile();
    public void SetImportFile( String sVal );
}

```

3.5.24 SpyTextView

```

public class SpyTextView
{
    public void ReleaseInstance();
    public SpyApplication GetApplication();
    public SpyDoc GetParent();
    public long LineFromPosition( long nCharPos );
    public long PositionFromLine( long nLine );
    public long LineLength( long nLine );
    public String GetSelText();
    public void SetSelText( String sText );
    public String GetRangeText( long nPosFrom, long nPosTill );
    public void ReplaceText( long nPosFrom, long nPosTill, String sText );
    public void MoveCaret( long nDiff );
    public void GoToLineChar( long nLine, long nChar );
    public void SelectText( long nPosFrom, long nPosTill );
    public long GetSelectionStart();
    public void SetSelectionStart( long nNewVal );
    public long GetSelectionEnd();
    public void SetSelectionEnd( long nNewVal );
    public String GetText();
    public void SetText( String sText );
    public long LineCount();
    public long Length();
}

```

3.5.25 SpyWSDL20DocumentationDlg

```
// Since version 2010
public class SpyWSDL20DocumentationDlg
{
    public void ReleaseInstance();
    public SpyApplication GetApplication();

    public long GetOptionsDialogAction();
    public void SetOptionsDialogAction( long nNewVal );

    public long GetOutputFileDialogAction();
    public void SetOutputFileDialogAction( long nNewVal );

    public boolean GetShowProgressBar();
    public void SetShowProgressBar( boolean bNewVal );

    public String GetOutputFile();
    public void SetOutputFile( String sNewVal );

    public long GetOutputFormat();
    public void SetOutputFormat( long nNewVal );

    public boolean GetMultipleOutputFiles();
    public void SetMultipleOutputFiles( boolean bNewVal );

    public boolean GetEmbedCSSInHTML();
    public void SetEmbedCSSInHTML( boolean bNewVal );

    public long GetDiagramFormat();
    public void SetDiagramFormat( long nNewVal );

    public boolean GetEmbedDiagrams();
    public void SetEmbedDiagrams( boolean bNewVal );

    public boolean GetCreateDiagramsFolder();
    public void SetCreateDiagramsFolder( boolean bNewVal );

    public boolean GetShowResult();
    public void SetShowResult( boolean bNewVal );

    public void IncludeAll( boolean bNewVal );
    public void AllDetails( boolean bNewVal );

    public boolean GetIncludeOverview();
    public void SetIncludeOverview( boolean bNewVal );

    public boolean GetIncludeService();
    public void SetIncludeService( boolean bNewVal );

    public boolean GetIncludeBinding();
    public void SetIncludeBinding( boolean bNewVal );

    public boolean GetIncludeInterface();
    public void SetIncludeInterface( boolean bNewVal );
}
```



```

    public boolean GetIncludeTypes();
    public void SetIncludeTypes( boolean bNewVal );

    public boolean GetIncludeImportedWSDLFiles();
    public void SetIncludeImportedWSDLFiles( boolean bNewVal );

    public boolean GetShowServiceDiagram();
    public void SetShowServiceDiagram( boolean bNewVal );

    public boolean GetShowBindingDiagram();
    public void SetShowBindingDiagram( boolean bNewVal );

    public boolean GetShowInterfaceDiagram();
    public void SetShowInterfaceDiagram( boolean bNewVal );

    public boolean GetShowTypesDiagram();
    public void SetShowTypesDiagram( boolean bNewVal );

    public boolean GetShowEndpoint();
    public void SetShowEndpoint( boolean bNewVal );

    public boolean GetShowSourceCode();
    public void SetShowSourceCode( boolean bNewVal );

    public boolean GetShowExtensibility();
    public void SetShowExtensibility( boolean bNewVal );

    public boolean GetShowUsedBy();
    public void SetShowUsedBy( boolean bNewVal );

    public boolean GetShowOperation();
    public void SetShowOperation( boolean bNewVal );

    public boolean GetShowFault();
    public void SetShowFault( boolean bNewVal );

    // Since version 2011r2
    public boolean GetUseFixedDesign();
    public void SetUseFixedDesign( boolean i_bVal );

    public String GetSPSFile();
    public void SetSPSFile( String i_strVal );
}

```

3.5.26 SpyWSDLDocumentationDlg

```

// Since version 2008r2sp1
public class SpyWSDLDocumentationDlg
{
    public void ReleaseInstance();
    public SpyApplication GetApplication();

    public String GetOutputFile();
    public void SetOutputFile( String sNewVal );

    public long GetOutputFileDialogAction();
    public void SetOutputFileDialogAction( long nNewVal );

    public long GetOptionsDialogAction();
}

```

```
public void SetOptionsDialogAction( long nNewVal );

public boolean GetShowProgressBar();
public void SetShowProgressBar( boolean bNewVal );

public boolean GetShowResult();
public void SetShowResult( boolean bNewVal );

public long GetOutputFormat();
public void SetOutputFormat( long nNewVal );

public boolean GetEmbedDiagrams();
public void SetEmbedDiagrams( boolean bNewVal );

public long GetDiagramFormat();
public void SetDiagramFormat( long nNewVal );

public boolean GetMultipleOutputFiles();
public void SetMultipleOutputFiles( boolean bNewVal );

public void IncludeAll( boolean bNewVal );

public boolean GetIncludeBinding();
public void SetIncludeBinding( boolean bNewVal );

public boolean GetIncludeImportedWSDLFiles();
public void SetIncludeImportedWSDLFiles( boolean bNewVal );

public boolean GetIncludeMessages();
public void SetIncludeMessages( boolean bNewVal );

public boolean GetIncludeOverview();
public void SetIncludeOverview( boolean bNewVal );

public boolean GetIncludePortType();
public void SetIncludePortType( boolean bNewVal );

public boolean GetIncludeService();
public void SetIncludeService( boolean bNewVal );

public boolean GetIncludeTypes();
public void SetIncludeTypes( boolean bNewVal );

public void AllDetails( boolean bNewVal );

public boolean GetShowBindingDiagram();
public void SetShowBindingDiagram( boolean bNewVal );

public boolean GetShowExtensibility();
public void SetShowExtensibility( boolean bNewVal );

public boolean GetShowMessageParts();
public void SetShowMessageParts( boolean bNewVal );

public boolean GetShowPort();
public void SetShowPort( boolean bNewVal );

public boolean GetShowPortTypeDiagram();
public void SetShowPortTypeDiagram( boolean bNewVal );

public boolean GetShowPortTypeOperations();
public void SetShowPortTypeOperations( boolean bNewVal );

public boolean GetShowServiceDiagram();
```

```

    public void SetShowServiceDiagram( boolean bNewVal );

    public boolean GetShowSourceCode();
    public void SetShowSourceCode( boolean bNewVal );

    public boolean GetShowTypesDiagram();
    public void SetShowTypesDiagram( boolean bNewVal );

    public boolean GetShowUsedBy();
    public void SetShowUsedBy( boolean bNewVal );

    // Since version 2010
    public boolean GetEmbedCSSInHTML();
    public void SetEmbedCSSInHTML( boolean i_bVal );

    public boolean GetCreateDiagramsFolder();
    public void SetCreateDiagramsFolder( boolean i_bVal );

    // Since version 2011r2
    public boolean GetUseFixedDesign();
    public void SetUseFixedDesign( boolean i_bVal );

    public String GetSPSFile();
    public void SetSPSFile( String i_strVal );

}

```

3.5.27 SpyXBRLDocumentationDlg

```

// Since version 2010
public class SpyXBRLDocumentationDlg
{
    public void ReleaseInstance();
    public SpyApplication GetApplication();

    public long GetOptionsDialogAction();
    public void SetOptionsDialogAction( long nNewVal );

    public long GetOutputDialogAction();
    public void SetOutputDialogAction( long nNewVal );

    public boolean GetShowProgressBar();
    public void SetShowProgressBar( boolean bNewVal );

    public String GetOutputFile();
    public void SetOutputFile( String sNewVal );

    public long GetOutputFormat();
    public void SetOutputFormat( long nNewVal );

    public boolean GetEmbedCSSInHTML();
    public void SetEmbedCSSInHTML( boolean bNewVal );

    public long GetDiagramFormat();
    public void SetDiagramFormat( long nNewVal );

    public boolean GetEmbedDiagrams();
    public void SetEmbedDiagrams( boolean bNewVal );
}

```

```
public boolean GetCreateDiagramsFolder();
public void SetCreateDiagramsFolder( boolean bNewVal );

public boolean GetShowResult();
public void SetShowResult( boolean bNewVal );

public void IncludeAll( boolean bNewVal );
public void AllDetails( boolean bNewVal );

public boolean GetIncludeOverview();
public void SetIncludeOverview( boolean bNewVal );

public boolean GetIncludeNamespacePrefixes();
public void SetIncludeNamespacePrefixes( boolean bNewVal );

public boolean GetIncludeGlobalElements();
public void SetIncludeGlobalElements( boolean bNewVal );

public boolean GetIncludeDefinitionLinkroles();
public void SetIncludeDefinitionLinkroles( boolean bNewVal );

public boolean GetIncludePresentationLinkroles();
public void SetIncludePresentationLinkroles( boolean bNewVal );

public boolean GetIncludeCalculationLinkroles();
public void SetIncludeCalculationLinkroles( boolean bNewVal );

public boolean GetShowDiagram();
public void SetShowDiagram( boolean bNewVal );

public boolean GetShowSubstitutiongroup();
public void SetShowSubstitutiongroup( boolean bNewVal );

public boolean GetShowItemtype();
public void SetShowItemtype( boolean bNewVal );

public boolean GetShowBalance();
public void SetShowBalance( boolean bNewVal );

public boolean GetShowPeriod();
public void SetShowPeriod( boolean bNewVal );

public boolean GetShowAbstract();
public void SetShowAbstract( boolean bNewVal );

public boolean GetShowNillable();
public void SetShowNillable( boolean bNewVal );

public boolean GetShowLabels();
public void SetShowLabels( boolean bNewVal );

public boolean GetShowReferences();
public void SetShowReferences( boolean bNewVal );

public boolean GetShowLinkbaseReferences();
public void SetShowLinkbaseReferences( boolean bNewVal );

public boolean GetShortQualifiedname();
public void SetShortQualifiedname( boolean bNewVal );

public boolean GetShowImportedElements();
```

```

    public void SetShowImportedElements( boolean bNewVal );

    // Since version 2011r2
    public boolean GetUseFixedDesign();
    public void SetUseFixedDesign( boolean i_bVal );

    public String GetSPSFile();
    public void SetSPSFile( String i_strVal );
};

```

3.5.28 SpyXMLData

```

public class SpyXMLData
{
    public void ReleaseInstance();
    public void AppendChild( SpyXMLData oNewData );
    public void EraseAllChildren();
    public void EraseCurrentChild();
    public SpyXMLData GetCurrentChild();
    public SpyXMLData GetFirstChild( long nKind );
    public SpyXMLData GetNextChild();
    public boolean GetHasChildren();
    public void InsertChild( SpyXMLData oNewData );
    public boolean IsSameNode( SpyXMLData oToComp);
    public long GetKind();
    public boolean GetMayHaveChildren();
    public String GetName();
    public void SetName( String sValue );
    public SpyXMLData GetParent();
    public String GetTextValue();
    public void SetTextValue( String sValue );
}

```

3.5.29 Authentic

SpyAuthenticRange

```

// Since version 2004R3
public class SpyAuthenticRange
{
    public void ReleaseInstance();
    public SpyApplication GetApplication();
    public SpyAuthenticView GetParent();
    public SpyAuthenticRange GotoNext( long eKind );
    public SpyAuthenticRange GotoPrevious( long eKind );
    public void Select();
    public long GetFirstTextPosition();
    public void SetFirstTextPosition( long nTextPosition );
    public long GetLastTextPosition();
    public void SetLastTextPosition( long nTextPosition );
    public String GetText();
    public void SetText( String strText );
    public boolean PerformAction( long eAction, String strElementName );
    public boolean CanPerformAction( long eAction, String strElementName );
    public String[] CanPerformActionWith( long eAction );
    public SpyAuthenticRange GoTo( long eKind, long nCount, long nFrom );
    public SpyAuthenticRange SelectNext( long eKind );
    public SpyAuthenticRange SelectPrevious( long eKind );
    public SpyAuthenticRange MoveBegin( long eKind, long nCount );
    public SpyAuthenticRange MoveEnd( long eKind, long nCount );
}

```

```

    public SpyAuthenticRange ExpandTo( long eKind );
    public SpyAuthenticRange CollapsToBegin();
    public SpyAuthenticRange CollapsToEnd();
    public SpyAuthenticRange GotoNextCursorPosition();
    public SpyAuthenticRange GotoPreviousCursorPosition();
    public boolean IsEmpty();
    public boolean IsEqual( SpyAuthenticRange ipCmp );
    public SpyAuthenticRange Clone();
    public SpyAuthenticRange SetFromRange( SpyAuthenticRange ipSrc );
    public boolean Delete();
    public boolean Cut();
    public boolean Copy();
    public boolean Paste();
    public SpyXMLData GetFirstXMLData();
    public void SetFirstXMLData( SpyXMLData objXMLDataPtr );
    public long GetFirstXMLDataOffset();
    public void SetFirstXMLDataOffset( long nOffset );
    public SpyXMLData GetLastXMLData();
    public void SetLastXMLData( SpyXMLData objXMLDataPtr );
    public long GetLastXMLDataOffset();
    public void SetLastXMLDataOffset( long nOffset );
    public String[] GetElementHierarchy();
    public String[] GetElementAttributeNames( String strElementName );
    public boolean HasElementAttribute( String strElementName, String
strAttributeName );
    public String GetElementAttributeValue( String strElementName, String
strAttributeName );
    public void SetElementAttributeValue( String strElementName, String
strAttributeName, String strNewValue );
    public String[] GetEntityNames();
    public void InsertEntity( String strEntityName );
    public boolean IsInDynamicTable();
    public boolean AppendRow();
    public boolean InsertRow();
    public boolean DuplicateRow();
    public boolean DeleteRow();
    public boolean MoveRowUp();
    public boolean MoveRowDown();

    // Since version 2004R4
    public boolean IsCopyEnabled();
    public boolean IsCutEnabled();
    public boolean IsPasteEnabled();
    public boolean IsDeleteEnabled();
    public boolean IsTextStateApplied( String i_strElementName );
    public boolean IsFirstRow();
    public boolean IsLastRow();
}

```

SpyAuthenticView

```

// Since version 2004R3
public class SpyAuthenticView
{
    public void ReleaseInstance();
    public SpyApplication GetApplication();
    public SpyDoc GetParent();
    public SpyAuthenticRange GetSelection();
    public void SetSelection( SpyAuthenticRange obj );
    public SpyAuthenticRange GetDocumentBegin();
    public SpyAuthenticRange GetDocumentEnd();
    public SpyAuthenticRange GetWholeDocument();
}

```

```

    public long GetMarkupVisibility();
    public void SetMarkupVisibility( long eSpyAuthenticMarkupVisibility );
    public SpyAuthenticRange GoTo( long eKind, long nCount, long nFrom );
    public void Print( boolean bWithPreview, boolean bPromptUser );
    public boolean Undo();
    public boolean Redo();
    public void UpdateXMLInstanceEntities();

    // Since version 2004R4
    public String GetAsXMLString();
    public void SetAsXMLString( String i_strXML );
    public SpyXMLData GetXMLDataRoot();
    public boolean IsUndoEnabled();
    public boolean IsRedoEnabled();
}

```

SpyDocEditSelection

```

public class SpyDocEditSelection
{
    public void ReleaseInstance();
    public SpyXMLData GetEnd();
    public long GetEndTextPosition();
    public SpyXMLData GetStart();
    public long GetStartTextPosition();
}

```

SpyDocEditView

```

public class SpyDocEditView
{
    public void ReleaseInstance();
    public void ApplyTextState( String sElementName );
    public SpyDocEditSelection GetCurrentSelection();
    public void EditClear();
    public void EditCopy();
    public void EditCut();
    public void EditPaste();
    public void EditRedo();
    public void EditSelectAll();
    public void EditUndo();
    public SpyXMLData GetNextVisible( SpyXMLData oElement );
    public SpyXMLData GetPreviousVisible( SpyXMLData oElement );
    public boolean GetIsEditClearEnabled();
    public boolean GetIsEditCopyEnabled();
    public boolean GetIsEditCutEnabled();
    public boolean GetIsEditPasteEnabled();
    public boolean GetIsEditRedoEnabled();
    public boolean GetIsEditUndoEnabled();
    public boolean GetIsRowAppendEnabled();
    public boolean GetIsRowDeleteEnabled();
    public boolean GetIsRowDuplicateEnabled();
    public boolean GetIsRowInsertEnabled();
    public boolean GetIsRowMoveDownEnabled();
    public boolean GetIsRowMoveUpEnabled();
    public boolean IsTextStateApplied( String sElementName );
    public boolean IsTextStateEnabled( String sElementName );
    public void LoadXML( String sXML );
    public void MarkupView( long nKind );
    public void RowAppend();
}

```

```

public void RowDelete();
public void RowDuplicate();
public void RowInsert();
public void RowMoveDown();
public void RowMoveUp();
public String SaveXML();
public void SelectionMoveTabOrder( boolean bForward, boolean bTag );
public boolean SelectionSet( SpyXMLData oStart, long nStartPos, SpyXMLData
oEndElement, long nEndPos );
public SpyXMLData GetXMLRoot();
public String[] GetAllowedElements( long nAction, SpyXMLData oStartPtr,
SpyXMLData oEndPtr );
}

```

3.5.30 Predefined constants

This section lists all classes that define the predefined constants used by the Java interface.

SPYApplicationStatus

```

public class SPYApplicationStatus
{
    public final static long spyApplicationStatus_Running           = 0;
    public final static long spyApplicationStatus_AfterLicenseCheck = 1;
    public final static long spyApplicationStatus_BeforeLicenseCheck = 2;
    public final static long spyApplicationStatus_ConcurrentLicenseCheckFailed = 3;
    public final static long spyApplicationStatus_ProcessingCommandLine = 4;
}

```

SPYAttributeTypeDefinition

```

public class SPYAttributeTypeDefinition
{
    public final static long spyMergedGlobal = 0;
    public final static long spyDistinctGlobal = 1;
    public final static long spyLocal = 2;
}

```

SPYAuthenticActions

```

public class SPYAuthenticActions
{
    public final static long spyAuthenticInsertAt = 0;
    public final static long spyAuthenticApply = 1;
    public final static long spyAuthenticClearSurr = 2;
    public final static long spyAuthenticAppend = 3;
}

```



```
    public final static long          = 4;
    spyAuthenticInsertBefore
    public final static long          = 5;
    spyAuthenticRemove
}
```

SPYAuthenticDocumentPosition

```
public class SPYAuthenticDocumentPosition
{
    public final static long          = 0;
    spyAuthenticDocumentBegin
    public final static long          = 1;
    spyAuthenticDocumentEnd
    public final static long          = 2;
    spyAuthenticRangeBegin
    public final static long          = 3;
    spyAuthenticRangeEnd
}
```

SPYAuthenticElementKind

```
public class SPYAuthenticElementKind
{
    public final static long          = 0;
    spyAuthenticChar
    public final static long          = 1;
    spyAuthenticWord
    public final static long          = 3;
    spyAuthenticLine
    public final static long          = 4;
    spyAuthenticParagraph
    public final static long          = 6;
    spyAuthenticTag
    public final static long          = 8;
    spyAuthenticDocument
    public final static long          = 9;
    spyAuthenticTable
    public final static long          = 10;
    spyAuthenticTableRow
    public final static long          = 11;
    spyAuthenticTableColumn
}
```

SPYAuthenticMarkupVisibility

```
public class SPYAuthenticMarkupVisibility
{
    public final static long          = 0;
    spyAuthenticMarkupHidden
    public final static long          = 1;
    spyAuthenticMarkupSmall
}
```

```

        public final static long spyAuthenticMarkupLarge = 2;
        public final static long spyAuthenticMarkupMixed = 3;
    }

```

SPYDatabaseKind

```

public class SPYLoading
{
    public final static long spyDB_Access = 0;
    public final static long spyDB_SQLServer = 1;
    public final static long spyDB_Oracle = 2;
    public final static long spyDB_Sybase = 3;
    public final static long spyDB_MySQL = 4;
    public final static long spyDB_DB2 = 5;
    public final static long spyDB_Other = 6;
    public final static long spyDB_Unspecified = 7;
}

```

SPYDialogAction

```

public class SPYDialogAction
{
    public final static long spyDialogOK = 0;
    public final static long spyDialogCancel = 1;
    public final static long spyDialogUserInput = 2;
}

```

SPYDOMType

```

public class SPYDOMType
{
    public final static long spyDOMType_msxml4 = 0;
    public final static long spyDOMType_xerces = 1;
}

```

SPYDTDSchemaFormat

```

public class SPYDTDSchemaFormat
{
    public final static long spyDTD = 0;
}

```

```
    public final static long spyDCD      = 1;
    public final static long spyXMLData = 2;
    public final static long spyBizTalk = 3;
    public final static long spyW3C     = 4;
}
```

SPYEncodingByteOrder

```
public class SPYEncodingByteOrder
{
    public final static long spyNONE      = 0;
    public final static long spyLITTLE_ENDIAN = 1;
    public final static long spyBIG_ENDIAN  = 2;
}
```

SPYExportNamespace

```
public class SPYExportNamespace
{
    public final static long spyNoNamespace      = 0;
    public final static long spyReplaceColonWithUnderscore = 1;
}
```

SPYFindInFilesSearchLocation

```
public class SPYFindInFilesSearchLocation
{
    public final static long spyFindInFiles_Documents = 0;
    public final static long spyFindInFiles_Project  = 1;
    public final static long spyFindInFiles_Folder   = 2;
}
```

SPYFrequentElements

```
public class SPYFrequentElements
{
    public final static long spyGlobalElements = 0;
    public final static long spyGlobalComplexType = 1;
}
```

SPYImageKind

```
public class SPYImageKind
{
    public final static long spyImageType_PNG = 0;
    public final static long spyImageType_EMF = 1;
}
```

```
}
```

SPYImportColumnsType

Enter topic text here.

SPYLibType

```
public class SPYLibType
{
    public final static long          = 0;
    spyLibType static
    public final static long          = 1;
    spyLibType_dll
}
```

SPYLoading

```
public class SPYLoading
{
    public final static long          = 0;
    spyUseCacheProxy
    public final static long          = 1;
    spyReload
}
```

SPYNumberDateTimeFormat

```
public class SPYNumberDateTimeFormat
{
    public final static long          = 0;
    spySystemLocale
    public final static long          = 1;
    spySchemaCompatible
}
```

SPYProgrammingLanguage

```
public class SPYLoading
{
    public final static long spyUndefinedLanguage = -1;
    public final static long spyJava              = 0;
    public final static long spyCpp                = 1;
    public final static long spyCSharp             = 2;
}
```

SPYProjectItemTypes

```
public class SPYProjectItemTypes
{
    public final static long          = 0;
    spyUnknownItem
}
```

```

    public final static long      = 1;
    spyFileItem
    public final static long      = 2;
    spyFolderItem
    public final static long      = 3;
    spyURLItem
}

```

SPYProjectType

```

public class SPYProjectType
{
    public final static long      = 0;
    spyVisualStudioProject
    public final static long      = 1;
    spyVisualStudio2003Project
    public final static long      = 2;
    spyBorlandProject
    public final static long      = 3;
    spyMonoMakefile
}

```

SPYSampleXMLGenerationOptimization

```

public class SPYSampleXMLGenerationOptimization
{
    public final static long      = 0;
    spySampleXMLGen_Optimized
    public final static long      = 1;
    spySampleXMLGen_NonMandatoryElements
    public final static long      = 2;
    spySampleXMLGen_Everything
}

```

SPYSampleXMLGenerationSchemaOrDTDAssignment

```

public class SPYSampleXMLGenerationOptimization
{
    public final static long      = 0;
    spySampleXMLGen_AssignRelatively
    public final static long      = 1;
    spySampleXMLGen_AssignAbsolutely
    public final static long      = 2;
    spySampleXMLGen_DoNotAssign
}

```

SPYSchemaDefKind

```

public class SPYSchemaDefKind
{
    public final static long spyKindElement = 0;
    public final static long      = 1;
    spyKindComplexType
    public final static long      = 2;
    spyKindSimpleType
}

```

```

    public final static long spyKindGroup      = 3;
    public final static long spyKindModel      = 4;
    public final static long spyKindAny        = 5;
    public final static long spyKindAttr       = 6;
    public final static long spyKindAttrGroup  = 7;
    public final static long spyKindAttrAny    = 8;
    public final static long spyKindIdentityUnique = 9;
    public final static long spyKindIdentityKey = 10;
    public final static long spyKindIdentityKeyRef = 11;
    public final static long spyKindIdentitySelector = 12;
    public final static long spyKindIdentityField = 13;
    public final static long spyKindNotation = 14;
    public final static long spyKindInclude = 15;
    public final static long spyKindImport = 16;
    public final static long spyKindRedefine = 17;
    public final static long spyKindFacet = 18;
    public final static long spyKindSchema = 19;
    public final static long spyKindCount = 20;
}

```

SPYSchemaDocumentationFormat

```

public class SPYSchemaDocumentationFormat
{
    public final static long spySchemaDoc_HTML = 0;
    public final static long spySchemaDoc_MSWord = 1;
    public final static long spySchemaDoc_RTF = 2;
    public final static long spySchemaDoc_PDF = 3;
}

```

SPYSchemaExtensionType

```

public class SPYSchemaExtensionType
{
    public final static long spySchemaExtension_None = 0;
    public final static long spySchemaExtension_SQL_XML = 1;
    public final static long spySchemaExtension_MS_SQL_Server = 2;
    public final static long spySchemaExtension_Oracle = 3;
}

```

SPYSchemaFormat

```
public class SPYSchemaFormat
{
    public final static long          = 0;
    spySchemaFormat Hierarchical
    public final static long          = 1;
    spySchemaFormat_Flat
}
```

SPYTextDelimiters

```
public class SPYTextDelimiters
{
    public final static long          = 0;
    spyTabulator
    public final static long          = 1;
    spySemicolon
    public final static long spyComma = 2;
    public final static long spySpace = 3;
}
```

SPYTextEnclosing

```
public class SPYTextEnclosing
{
    public final static long          = 0;
    spyNoEnclosing
    public final static long          = 1;
    spySingleQuote
    public final static long          = 2;
    spyDoubleQuote
}
```

SPYTypeDetection

```
public class SPYTypeDetection
{
    public final static long          = 0;
    spyBestPossible
    public final static long          = 1;
    spyNumbersOnly
    public final static long          = 2;
    spyNoDetection
}
```

SPYURLTypes

```
public class SPYURLTypes
{
    public final static long          = (-1);
    spyURLTypeAuto
    public final static long          = 0;
    spyURLTypeXML
}
```

```

        public final static long          = 1;
        spyURLTypeDTD
    }

```

SpyViewModes

```

public class SPYViewModes
{
    public final static long          = 0;
    spyViewGrid
    public final static long          = 1;
    spyViewText
    public final static long          = 2;
    spyViewBrowser
    public final static long          = 3;
    spyViewSchema
    public final static long          = 4;
    spyViewContent
    public final static long          = 4;
    spyViewAuthentic
    public final static long          = 5;
    spyViewWSDL
    public final static long spyViewZIP = 6;
    public final static long          = 7;
    spyViewEditionInfo
}

```

SPYWhitespaceComparison

```

public class SPYWhitespaceComparison
{
    public final static long          = 0;
    spyCompareAsIs
    public final static long          = 1;
    spyCompareNormalized
    public final static long          = 2;
    spyStripAll
}

```

SPYXMLDataKind

```

public class SPYXMLDataKind
{
    public final static long          = 0;
    spyXMLDataXMLDocStruct
    public final static long          = 1;
    spyXMLDataXMLEntityDocStruct
    public final static long          = 2;
    spyXMLDataDTDDocStruct
    public final static long          = 3;
    spyXMLDataXML
    public final static long          = 4;
    spyXMLDataElement
    public final static long          = 5;
    spyXMLDataAttr
}

```



```
    public final static long          = 6;
    spyXMLDataText
    public final static long          = 7;
    spyXMLDataCData
    public final static long          = 8;
    spyXMLDataComment
    public final static long          = 9;
    spyXMLDataPI
    public final static long          = 10;
    spyXMLDataDefDoctype
    public final static long          = 11;
    spyXMLDataDefExternalID
    public final static long          = 12;
    spyXMLDataDefElement
    public final static long          = 13;
    spyXMLDataDefAttlist
    public final static long          = 14;
    spyXMLDataDefEntity
    public final static long          = 15;
    spyXMLDataDefNotation
    public final static long          = 16;
    spyXMLDataKindsCount
}
```

4 ActiveX Integration

XMLSpyControl is a control that provides a means of integration of the XMLSpy user interface and the functionality described in this section into most kinds of applications. ActiveX technology was chosen so as to allow integration using any of a wide variety of languages; this enables C++, C#, VisualBasic, or HTML to be used for integration. ActiveX components officially only work with Microsoft Internet Explorer. All components are full OLE Controls, which makes integration as simple as possible. Two different levels of integration are provided, thus enabling the integration to be adapted to a wide range of needs.

To integrate XMLSpy you must install the XMLSpy Integration Package. Ensure that you install XMLSpy first, and then the XMLSpy Integration Package.

For a successful integration you have to consider the following main design factors:

- What technology or programming language can the hosting application use to integrate the XMLSpyControl?
- Should the integrated UI look exactly like XMLSpy with all its menus, toolbars, and windows, or will a subset of these elements—like allowing only one document and a restricted set of commands—be more effective?
- How deep will the integration be? Should the XMLSpy user interface be used as is? Are user interface extensions and/or restrictions required? Can some frequently used tasks be automated?

The sections, [Integration at the Application Level](#) and [Integration at Document Level](#), both of which have examples in various programming languages, will help you to make the right decisions quickly. The section, [Object Reference](#), describes all COM objects that can be used for integration, together with their properties and methods.

For automation tasks, the [XMLSpy Automation Interface](#) is accessible from the XMLSpyControl as well.

For information about how to integrate XMLSpy into Microsoft Visual Studio see the section, [XMLSpy in Visual Studio](#).

4.1 Integration at Application Level

Integration at application level is simple and straightforward. It allows you to embed the complete interface of XMLSpy into a window of your application. Since you get the whole user interface of XMLSpy, you get all menus, toolbars, the status bar, document windows, and helper windows. Customization of the application's user interface is restricted to what XMLSpy provides. This includes rearrangement and resizing of helper windows and customization of menus and toolbars.

The only ActiveX control you need to integrate is [XMLSpyControl](#). Its property [IntegrationLevel](#) defaults to application-level. You may use [Appearance](#) and [BorderStyle](#) to configure the appearance of the control's wrapper window. Do not instantiate or access [XMLSpyControlDocument](#) or [XMLSpyControlPlaceHolder](#) ActiveX controls when integrating at application-level.

If you have any initialization to do or if you want to automate some behaviour of XMLSpy, use the properties, methods, and events described for [XMLSpyControl](#). Consider using [XMLSpyControl.Application](#) for more complex access to XMLSpy functionality.

In this section is an example ([Example: HTML](#)) showing how the XMLSpy application can be embedded in an HTML page. For usage with other programming languages, or more sophisticated access, see the [Examples](#) of integration at document-level.

4.1.1 Example: HTML

This example shows a simple integration of the XMLSpy control at application-level into a HTML page. The integration is described in the following sections:

- Instantiate a XMLSpyControl in HTML code.
- Implement buttons to load documents and automate code-generation tasks.
- Define actions for some application events.

The code for this example is available at the following location in your XMLSpy installation:
Examples\ActiveX\HTML\XMLSpyActiveX_ApplicationLevel.htm

Note: This example works only in Internet Explorer.

Instantiate the Control

The HTML `Object` tag is used to create an instance of the XMLSpyControl. The `Classid` is that of XMLSpyControl. Width and height specify the window size. No additional parameters are necessary, since application-level is the default.

```
<OBJECT id="objXMLSpyControl"
  Classid="clsid: a258bba2-3835-4c16-8590-72b44f52c471"
  width="1000"
  height="700"
  VIEWASTEXT>
</OBJECT>
```

Add Button to Open Default Document

As a simple example of how to automate some tasks, we add a button to the page:

```
<input type="button" value="Open Marketing Expenses"
```

```
onclick="BtnOpenMEFile()">
```

When clicked, a predefined document will be opened in the XMLSpyControl. We use a method to locate the file relative to the XMLSpyControl so the example can run on different installations.

```
<SCRIPT ID=Javahandlers LANGUAGE=javascript>
// -----
// open a pre-defined document
function BtnOpenMEFile()
{
    objXMLSpyControl.Open("C:\Documents and Settings\username\My
Documents\Altova\XMLSpy2012\Examples\OrgChart.xml");
}
</SCRIPT>
```

Add Buttons for Code Generation

For direct access, we want to have a button that will validate the current document. The method is similar to that used in the previous section.

First comes the button:

```
<input type="button" value="Validate" onclick="BtnValidate()">
```

Then we provide the script that will validate the current document.

```
<SCRIPT ID=Javahandlers LANGUAGE=javascript>
// -----
// check validity of current document.
// if validation fails, show validation result in alert box .
function BtnValidate()
{
    // get top-level object of automation interface
    var objApp = objXMLSpyControl.Application;

    // get the active document
    var objDocument = objApp.ActiveDocument;

    if ( objDocument == null )
        alert( "no active document found" );
    else
    {
        // define as arrays to support their usage as return parameters
        var errorText = new Array(1);
        var errorPos = new Array(1);
        var badData = new Array(1);

        var valid = objDocument.IsValid(errorText, errorPos, badData);

        if (! valid)
        {
            // compose the error description
            var text = errorText;

            // access that XMLData object only if filled in
            if (badData[0] != null)
                text += "(" + badData[0].Name + "/" +
badData[0].TextValue + ")";

            alert("Validation error[" + errorPos + "]: " + text);
        }
    }
}
```

```
        }
        else
            alert("Docuent is valid"); }
    }
</SCRIPT>
```

Connect to Custom Events

The example implements two event callbacks for XMLSpyControl custom events to show the principle:

```
<!-- ----- -->
<!-- custom event 'OnDocumentOpened' of XMLSpyControl object -->
<SCRIPT LANGUAGE="javascript">
    function objXMLSpyControl::OnDocumentOpened( objDocument )
    {
        // alert("Document '" + objDocument.Name + "' opened!");
    }
</SCRIPT>

<!-- ----- -->
<!-- custom event 'OnDocumentClosed' of XMLSpyControl object -->
<SCRIPT LANGUAGE="javascript">
    function objXMLSpyControl::OnDocumentClosed( objDocument )
    {
        // alert("Document '" + objDocument.Name + "' closed!");
    }
</SCRIPT>
```

4.2 Integration at Document Level

Integration at document level gives you freedom over instantiation and placement of the following parts of the XMLSpy user interface:

- Editing windows for XMLSpy
- XMLSpy entry helper windows
- XMLSpy validator output window
- XMLSpy project window
- XMLSpy XPath profiler window
- XMLSpy XPath dialog window
- XMLSpy XSLT/XQuery debugger windows
- XMLSpy SOAP debugger window

If necessary, a replacement for the menus and toolbars of XMLSpy must be provided by your application.

You will need to instantiate and access multiple ActiveX controls, depending on which user interface parts you want to re-use. All these controls are contained in the XMLSpyControl OCX.

- [Use XMLSpyControl](#) to set the integration level and access application wide functionality.
- [Use XMLSpyControlDocument](#) to create any number of editor windows. It may be sufficient to create only one window and re-use it, depending on your needs.
- Optionally [Use XMLSpyControlPlaceholder](#) to embed XMLSpy entry helper windows, validator output or other windows mentioned above.
- Access run-time information about commands, menus, and toolbars available in XMLSpyControl to seamlessly integrate these commands into your application's menus and toolbars. See [Query XMLSpy Commands](#) for more information.

If you want to automate some behaviour of XMLSpy use the properties, methods, and events described for the [XMLSpyControl](#), [XMLSpyControlDocument](#) and [XMLSpyControlPlaceholder](#). Consider using [XMLSpyControl.Application](#), [XMLSpyControlDocument.Document](#) and [XMLSpyControlPlaceholder.Project](#) for more complex access to XMLSpy functionality. However, to open a document always use [XMLSpyControlDocument.Open](#) or [XMLSpyControlDocument.New](#) on the appropriate document control. To open a project always use [XMLSpyControlPlaceholder.OpenProject](#) on a placeholder control embedding a XMLSpy project window.

See [Examples](#) on how to instantiate and access the necessary controls in different programming environments.

4.2.1 Use XMLSpyControl

To integrate at document level, instantiate a [XMLSpyControl](#) first. Set the property [IntegrationLevel](#) to `ActiveXIntegrationOnDocumentLevel(=1)`. Set the window size of the embedding window to `0x0` to hide any user interface behind the control. You may use [Appearance](#) and [BorderStyle](#) to configure the appearance of the control's wrapper window.

Avoid using the method [Open](#) since this might lead to unexpected results. Use the corresponding open methods of [XMLSpyControlDocument](#) and [XMLSpyControlPlaceholder](#), instead.

See [Query XMLSpy Commands](#) for a description of how to integrate XMLSpy commands into your application. Send commands to XMLSpy via the method [Exec](#). Query if a command is currently enabled or disabled using the method [QueryStatus](#).

4.2.2 Use XMLSpyControlDocument

An instance of the `XMLSpyControlDocument` ActiveX control allows you to embed one XMLSpy document editing window into your application. You can use any number of instances you need.

Use the method [Open](#) to load any other existing file.

The control does not support a read-only mode. The value of the property [ReadOnly](#) is ignored.

Use [Path](#) and [Save](#) or methods and properties accessible via the property [Document](#) to access document functionality.

4.2.3 Use XMLSpyControlPlaceholder

Instances of `XMLSpyControlPlaceholder` ActiveX controls allow you to selectively embed the additional helper windows of XMLSpy into your application. The property [PlaceholderWindowID](#) selects the XMLSpy helper window to be embedded. Use only one `XMLSpyControlPlaceholder` for each window identifier. See [Enumerations.XMLSpyControlPlaceholderWindow](#) for valid window identifiers.

For placeholder controls that select the XMLSpy project window, additional methods are available. Use [OpenProject](#) to load a XMLSpy project. Use the property [Project](#) and the methods and properties from the XMLSpy automation interface to perform any other project related operations.

4.2.4 Query XMLSpy Commands

When integrating at document-level, no menu or toolbar from XMLSpy is available to your application. Instead, you can query all the commands and the structure of the application menu at runtime. Professional applications will need to integrate this menu in a sophisticated manner into their own menu structure. Your installation of XMLSpy even provides you with command label images used within XMLSpy. See the folder `Examples\ActiveX\Images` of your XMLSpy installation for icons in GIF format. The file names correspond to the [labels](#) of commands.

4.2.5 Examples

This section contains examples of XMLSpy document-level integration using different container environments and programming languages. Source code for all examples is available in the folder `Examples\ActiveX` of your XMLSpy installation.

C#

The C# example shows how to integrate the XMLSpyControl in a common desktop application created with C# using Visual Studio 2008. The following topics are covered:

- Integration of a `XMLSpyControl Document` control to embed a XMLSpy document editing window.
- Usage of a `XMLSpyControlPlaceholder` controls to embed the XMLSpy XPath dialog window.
- Opening and closing of XMLSpy documents via the main menu.

Please note that the example application is already complete. There is no need to change anything if you want to run and see it working. The following steps describe what general actions and considerations must be taken in order to create a project such as this.

Introduction

Adding the XMLSpy components to the Toolbox

Before you take a look at the sample project please add the assemblies to the .NET IDE Toolbox. The XMLSpy Installer will have already installed the assemblies in the .NET Global Assembly Cache (GAC). If you open the Toolbox dialog under **Tools | Add/Remove Toolbox Items** the controls will appear as `AxXMLSpyControl`, `AxXMLSpyControlDocument` and `AxXMLSpyControlPlaceholder` on the .NET Framework Components tab. Check all to make them available to the IDE.

Now you can open the `XPathDialog.sln` file in the `ActiveX\C#\XPathDialog` folder to load the project.

Placing the XMLSpyControl

It is necessary to have one `XMLSpyControl` instance to set the integration level and to manage the Document and Placeholder controls of the XMLSpy library. The control is accessible via the General section of the Toolbox helper window in the IDE. To add it you need to select the component in the Toolbox window and drag a rectangle wherever you want to have it in the destination window. If you have an application which does not open a window on startup you can use a simple invisible Form with the control on it which is created manually in the code.

The example project adds this instance to the main MdiContainer `MDIMain`. If you open `MDIMain` in the Design View from the Solution Explorer you will see a light blue rectangle at the top-left side in the client area of the Frame window. Selecting this rectangle will show you the properties of the `XMLSpyControl`. It is important to set the `IntegrationLevel` property to `ICActiveXIntegrationOnDocumentLevel` in order to turn on the Document and Placeholder support of the XMLSpy library. You can set the `Visible` flag to `False` to avoid any confusion about the control for the user.

Adding the Document Control

Document editing window on the MDI Frame

The example project uses one Document control show and edit one XMLSpy document in the main MDI Frame. It is added via the Toolbox window by dragging a rectangle on the destination Form. The Document control also has the `Anchor` and `Dock` properties set in order to react on resizing of the Frame window.

Adding the Placeholder Control

Placeholders on the MDI Frame

The example project uses one Placeholder control to embed the XPath Dialog window into the main MDI Frame. It is added via the Toolbox window by dragging a rectangle on the destination Form. To set the type of the Placeholder which should be displayed one has to set the `PlaceholderWindowID` property. This property can also be changed during runtime in the code of the application. The Placeholder control would change its content immediately. The Placeholder also has the `Anchor` properties set in order to react on resizing of the Frame window.

Retrieving Command Information

The XMLSpyControl gives access to all commands of XMLSpy through its `CommandsStructure` property. The example project uses the [XMLSpyCommands](#) and [XMLSpyCommand](#) interfaces to dynamically build a menu in the MDI Frame window.

The code to add the commands will be placed in the `MDIMain` method of the `XMLSpyApplication` class in the file `MDIMain.cs`:

```
public MDIMain()
{
    .
    .
    .
    XMLSpyControlLib.XMLSpyCommands objCommands;
    objCommands = axXMLSpyControl.CommandsStructure;

    long nCount = objCommands.Count;

    for(long idx = 0; idx < nCount; idx++)
    {
        XMLSpyControlLib.XMLSpyCommand objCommand;
        objCommand = objCommands[(int)idx];

        // We are looking for the Menu with the name IDR_XMLSPY. This menu
        // contains
        // the complete main menu of XMLSpy.

        if(objCommand.Label == "IDR_XMLSPY")
        {
            InsertMenuStructure(mainMenu.MenuItems, 1, objCommand, 0, 0,
                               false);
        }
    }
    .
    .
    .
}
```

`mainMenu` is the name of the menu object of the MDI Frame window created in the Visual Studio IDE. `InsertMenuStructure` takes the XMLSpy menu from the `IDR_XMLSPY` command object and adds the XMLSpy menu structure to the already existing menu of the sample project. No commands from the **File**, **Project**, or **Window** menu are added.

The new commands are instances of the class `CustomMenuItem`, which is defined in `CustomMenuItem.cs`. This class has an additional member to save the XMLSpy command ID, which is taken to execute the command using [Exec](#) on selecting the menu item. This code from `InsertMenuStructure` creates the new command:

```
CustomMenuItem newMenuItem = new CustomMenuItem();

if(objCommand.IsSeparator)
    newMenuItem.Text = "-";
else
{
    newMenuItem.Text = strLabel;
    newMenuItem.m_XMLSpyCmdID = (int)objCommand.ID;
    newMenuItem.Click += new EventHandler(AltovaMenuItem_Click);
}
```

You can see that all commands get the same event handler `AltovaMenuItem_Click` which does the processing of the command:

```
private void AltovaMenuItem_Click(object sender, EventArgs e)
{
    if(sender.GetType() == System.Type.GetType("XMLSpy
Application.CustomMenuItem"))
    {
        CustomMenuItem customItem = (CustomMenuItem) sender;

        ProcessCommand(customItem.m_XMLSpyCmdID);
    }
}

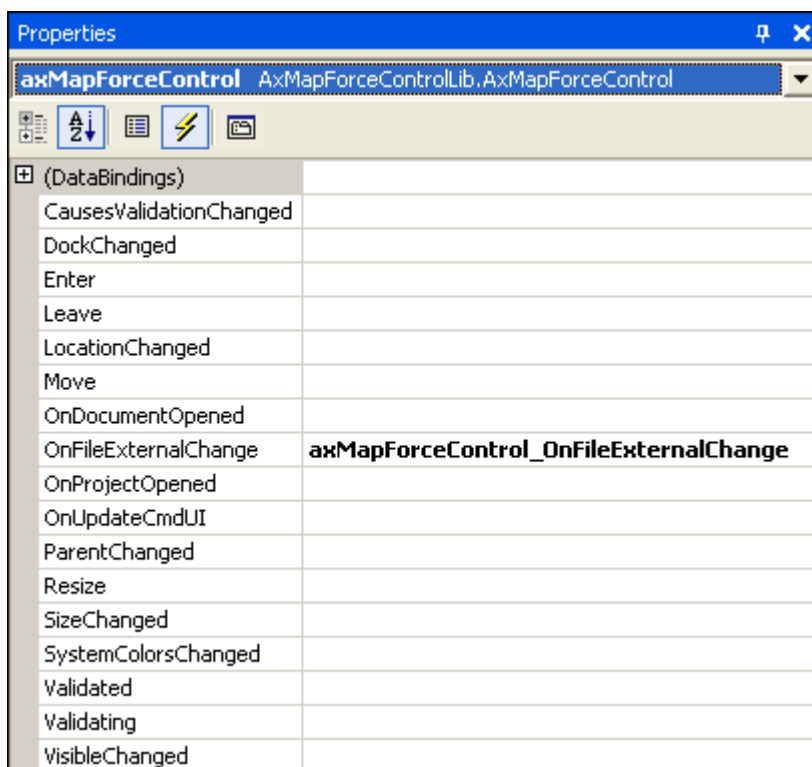
private void ProcessCommand(int nID)
{
    XMLSpyDoc docXMLSpy = GetCurrentXMLSpyDoc();

    if(docXMLSpy != null)
        docXMLSpy.axXMLSpyControlDoc.Exec(nID);
    else
        axXMLSpyControl.Exec(nID);
}
```

`ProcessCommand` delegates the execution either to the `XMLSpyControl` itself or to any active `XMLSpy` document loaded in a `XMLSpyControlDocument` control. This is necessary because the `XMLSpyControl` has no way to know which document is currently active in the hosting application.

Handling Events

Because all events in the `XMLSpy` library are based on connection points, you can use the C# delegate mechanism to provide the custom event handlers. You will always find a complete list of events on the property page of each control of the `XMLSpy` library. The picture below shows the events of the main `XMLSpyControl`:



As you can see, the example project only overrides the `OnFileExternalChange` event. The creation of the C# delegate is done for you by the C# Framework. All you need to do is to fill the empty event handler. The handler implementation turns off any file reloading and displays a message box to inform the user that a file loaded by the XMLSpyControl has been changed from outside:

```
private void axMapForceControl_OnFileExternalChange(object sender,
AxMapForceControlLib._DMapForceControlEvents_OnFileExternalChangeEvent e)
{
    MessageBox.Show("Attention: The file " + e.strPath + " has been changed
from outside\nbut reloading is turned off in the sample application!");

    // This turns off any file reloading:
    e.varRet = false;
}
```

Testing the Example

After adding the assemblies to the Toolbox (see [Introduction](#)), you can run the sample project with F5 without the need to change anything in the code. The main MDI Frame window is created and contains an editing window with an empty XML document and a XPath Dialog window of XMLSpy at the bottom. Use **File | Open** to open the file `OrgChart.xml`, which is in the XMLSpy examples folder. The file is loaded and displayed.

After you load the document, you can try using the XPath dialog.

HTML

This example shows an integration of the XMLSpy control at document-level into a HTML page. The following topics are covered:

- Instantiate a XMLSpyControl ActiveX control object in HTML code
- Instantiate a XMLSpyControlDocument ActiveX control to allow editing a XMLSpy file
- Instantiate one XMLSpyControlPlaceHolder for a XMLSpyControl project window
- Instantiate one XMLSpyControlPlaceHolder to alternatively host one of the XMLSpy helper windows
- Instantiate one XMLSpyControlPlaceHolder ActiveX control to show the XMLSpy validation output window
- Create a simple customer toolbar for some heavy-used XMLSpy commands
- Add some more buttons that use the COM automation interface of XMLSpy
- Use event handlers to update command buttons

This example is available in its entirety in the file `XMLSpyActiveX_ApplicationLevel.htm` within the `C:\Program Files\Altova\XMLSpy2012\Examples\ActiveX\HTML\` folder of your XMLSpy installation.

Note: This example works only in Internet Explorer.

Instantiate the XMLSpyControl

XMLSpyControlThe HTML `OBJECT` tag is used to create an instance of the XMLSpyControl. The Classid is that of XMLSpyControl. Width and height are set to 0 since we use this control as manager control without use for its user interface. The integration level is specified as a parameter within the `OBJECT` tag.

```
<OBJECT id="objXMLSpyXXMLSpyControl"
        Classid="clsid: a258bba2-3835-4c16-8590-72b44f52c471"
        width="0"
        height="0"
        VIEWASTEXT>
    <PARAM NAME="IntegrationLevel" VALUE="1">
</OBJECT>
```

Create Editor Window

The HTML `OBJECT` tag is used to embed an editing window.

```
<OBJECT id="objDoc1"
        Classid="clsid: DFBB0871-DAFE-4502-BB66-08CEB7DF5255"
        width="600"
        height="500"
        VIEWASTEXT>
</OBJECT>
```

Create Project Window

The HTML `OBJECT` tag is used to create a XMLSpyControlPlaceHolder window. The first additional custom parameter defines the placeholder to show the XMLSpy project window. The second parameter loads one of the example projects delivered with your XMLSpy installation

(located in the <yourusername>/MyDocuments folder).

```
<OBJECT id="objProjectWindow"
  Classid="clsid: FDEC3B04-05F2-427d-988C-F03A85DE53C2"
  width="200"
  height="200"
  VIEWASTEXT>
  <PARAM name="PlaceholderWindowID" value="3">
  <PARAM name="FileName" value="Examples/Examples.spp">
</OBJECT>
```

Create Placeholder for Helper Windows

The HTML OBJECT tag is used to instantiate a XMLSpyControlPlaceHolder ActiveX control that can host the different XMLSpy helper windows. Initially, no helper window is shown. See the example file.

```
<OBJECT id="objEHWindow"
  Classid="clsid: 135DEEF4-6DF0-47c2-8F8C-F145F5F3F672"
  width="200"
  height="200"
  VIEWASTEXT>
  <PARAM name="PlaceholderWindowID" value="0">
</OBJECT>
```

Three buttons allow us to switch the actual window that will be shown. The JavaScript execute on-button-click sets the property PlaceholderWindowID to the corresponding value defined in [XMLSpyControlPlaceholderWindow](#).

```
<input type="button" value="EH - Elements" onclick="BtnHelperWindow( 0) ">
<input type="button" value="EH - Attributes" onclick="BtnHelperWindow( 1) ">
<input type="button" value="EH - Entities" onclick="BtnHelperWindow( 2) ">
```

```
<SCRIPT ID="Javahandlers" LANGUAGE="javascript">
//
-----
// specify which of the windows shall be shown in the placeholder control.
function BtnHelperWindow(i_ePlaceholderWindowID)
{
  objEHWindow.PlaceholderWindowID = i_ePlaceholderWindowID;
}
</SCRIPT>
```

Create a Custom Toolbar

The custom toolbar consists of buttons with images of XMLSpy commands. The command ID numbers can be found in the button elements shown below [Command Table](#).

```
<button id="btnWellFormed" title="Check Well-formedness"
  onclick="BtnDoCommand( 34049) ">
  
</button>
<button id="btnValidate" title="Validate" onclick="BtnDoCommand( 34174) ">
  
</button>
```

On clicking one of these buttons the corresponding command ID is sent to the manager control.

```

<SCRIPT ID="Javahandlers" LANGUAGE="javascript">
// -----
// perform any command specified by cmdID.
// command routing includes application, active document and view.
function BtnDoCommand( cmdID)
{
    objXMLSpyX.Exec( cmdID );
    msgtext.innerText = "Command " + cmdID + " performed.";
}
</SCRIPT>

```

Create More Buttons

In the example, we add some more buttons to show some automation code.

```

<p>
  <input type="button" value="New File" onclick="BtnNewFile( objDoc1) ">
  <input type="button" value="Save File" onclick="BtnSaveFile( objDoc1) ">
  <input type="text" title="Path" id="strPath" width="150">
  <input type="button" value="Open OrgChart.xml"
onclick="BtnOpenFile( objDoc1, ' OrgChart.xml' )">
  <input type="button" value="Open OrgChart.xsd"
onclick="BtnOpenFile( objDoc1, ' OrgChart.xsd' )">
<p>

```

The corresponding JavaScript looks like this:

```

<SCRIPT ID="Javahandlers" LANGUAGE="javascript">
// -----
// open a document in the specified document control window.
function BtnOpenFile( objDocCtrl, strFileName)
{
    // do not use XMLSpyX.Application.OpenDocument(...) to open a document,
    // since then XMLSpyControl wouldn't know a control window to show
    // the document in. Instead:

    objDocCtrl.Open( "C:\Documents and Settings\username\My
Documents\Altova\XMLSpy2012\Examples/" + strFileName);
    objDocCtrl.setActive();
}

// -----
// open a new empty document in the specified document control window.
function BtnNewFile( objDocCtrl)
{
    objDocCtrl.Open( "" );
    objDocCtrl.setActive();
}

// -----
// Saves the current file in the specified document control window.
function BtnSaveFile( objDocCtrl)
{
    if( objDocCtrl.Path.length > 0)
        objDocCtrl.SaveDocument();
    else
    {
        if( strPath.value.length > 0)
        {
            objDocCtrl.Path = strPath.value;
            objDocCtrl.Save();
        }
        else

```

```

        {
            alert("Please set path for the document first!");
            strPath.focus();
        }
    }

    objDocCtrl.SetActive();
}

// -----
// check validity of current document.
// if validation fails, show validation result in alert box .
function BtnValidate()
{
    // get top-level object of automation interface
    var objApp = objXMLSpyControl.Application;

    // get the active document
    var objDocument = objApp.ActiveDocument;

    if ( objDocument == null )
        alert( "no active document found" );
    else
    {
        // define as arrays to support their usage as return parameters
        var errorText = new Array(1);
        var errorPos = new Array(1);
        var badData = new Array(1);

        var valid = objDocument.IsValid(errorText, errorPos, badData);

        if (! valid)
        {
            // compose the error description
            var text = errorText;

            // access that XMLData object only if filled in
            if (badData[0] != null)
                text += "(" + badData[0].Name + "/" +
badData[0].TextValue + ")";

            alert("Validation error[" + errorPos + "]: " + text);
        }
        else
            alert("Docuent is valid");
    }
}
</SCRIPT>

```

Create Event Handler to Update Button Status

Availability of a command may vary with every mouseclick or keystroke. The custom event `OnUpdateCmdUI` of `XMLSpyControl` gives us an opportunity to update the enabled/disabled state of buttons associated with XMLSpy commands. The method [XMLSpyControl.QueryStatus](#) is used to query whether a command is enabled or not.

```

<SCRIPT LANGUAGE="javascript">

    function objXMLSpyX::OnUpdateCmdUI()
    {
        if ( document.readyState == "complete" )           // 'complete'
        {

```

```
        // update status of buttons  
        btnWellFormed.disabled = ! ( objDoc1.QueryStatus( 34049) & 0x02);  
    // not enabled  
        btnValidate.disabled = ! ( objDoc1.QueryStatus( 34174) & 0x02);  
    // not enabled  
    }  
    }  
  
    // set activity status of simulated toolbar  
</SCRIPT>
```


4.3 Command Table for XMLSpy

Tables in this section list the names and identifiers of all commands that are available within XMLSpy. Every sub-section lists the commands from the corresponding top-level menu of XMLSpy. The left-most column shows the command's menu text to make it easier for you to identify the functionality behind the command. The last sub-section is a collection of those commands that are not accessible via the main menu.

Depending on the edition of XMLSpy you have installed, some of these commands might not be supported. See [Query XMLSpy Commands](#) on how to query the current resource structure and command availability.

Use the command identifiers with [XMLSpyControl.QueryStatus](#) or [XMLSpyControlDocument.QueryStatus](#) to check the current status of a command. Use [XMLSpyControl.Exec](#) or [XMLSpyControlDocument.Exec](#) to execute a command.

[File Menu](#)
[Edit Menu](#)
[Project Menu](#)
[XML menu](#)
[DTD/Schema Menu](#)
[Schema Design Menu](#)
[XSL/XQuery Menu](#)
[Authentic Menu](#)
[Convert Menu](#)
[View Menu](#)
[Browser Menu](#)
[WSDL Menu](#)
[SOAPMenu](#)
[Tools Menu](#)
[Window Menu](#)
[Help Menu](#)

4.3.1 File Menu

Commands from the File menu:

Menu Text	Command Name	ID
New...	ID_FILE_NEW	57600
Open...	ID_FILE_OPEN	57601
Reload	IDC_FILE_RELOAD	34065
Encoding...	IDC_ENCODING	34061
Close	ID_FILE_CLOSE	57602
Close All	IDC_CLOSE_ALL	34050
Save	ID_FILE_SAVE	57603
Save As...	ID_FILE_SAVE_AS	57604
Save to URL...	ID_FILE_SAVE_TO_URL	34209
Save All	ID_FILE_SAVE_ALL	34208
Send by Mail...	ID_FILE_SEND_MAIL	57612
Print...	IDC_PRINT	34103
Print Preview	IDC_PRINT_PREVIEW	34104
Print Setup...	ID_FILE_PRINT_SETUP	57606

Recent File	ID_FILE_MRU_FILE1	57616
Exit	ID_APP_EXIT	57665

4.3.2 Edit Menu

Commands from the Edit menu:

Menu Text	Command Name	ID
Undo	ID_EDIT_UNDO	57643
Redo	ID_EDIT_REDO	57644
Cut	ID_EDIT_CUT	57635
Copy	ID_EDIT_COPY	57634
Paste	ID_EDIT_PASTE	57637
Delete	ID_EDIT_CLEAR	57632
Copy as XML-Text	IDC_COPY_AS_XML_TEXT	33443
Copy as Structured text	IDC_COPY_AS_STRUCTURED_TEXT	33442
Copy XPath	IDC_COPY_XPATH	33444
Pretty-Print XML Text	IDC_PRETTY_PRINT	34101
Select All	ID_EDIT_SELECT_ALL	57642
Find...	ID_EDIT_FIND	57636
Find next	ID_EDIT_REPEAT	57640
Replace...	ID_EDIT_REPLACE	57641
Find in files...	IDC_FIND_IN_FILES	34000
Insert/Remove Bookmark	IDC_TOGGLE_BOOKMARK	34162
Remove All Bookmarks	IDC_REMOVEALLBOOKMARKS	34132
Goto Next Bookmark	IDC_GOTONEXTBOOKMARK	34070
Goto Previous Bookmark	IDC_GOTOPREVBOOKMARK	34071

4.3.3 Project Menu

Commands from the Project menu:

Menu Text	Command Name	ID
New Project	IDC_PROJECT_NEW	34122
Open Project...	IDC_PROJECT_OPEN	34123
Reload Project	IDC_PROJECT_RELOAD	34126
Close Project	IDC_PROJECT_CLOSE	34113
Save Project	IDC_PROJECT_SAVE	34127
Source Control/Open Project...	IDC_SCC_OPEN_PROJECT	34140
Source Control/Enable Source Code Control	IDC_SCC_ENABLE	34137

Source Control/Get latest version...	IDC_SCC_GET	34138
Source Control/Check Out...	IDC_SCC_CHECK_OUT	34135
Source Control/Check In...	IDC_SCC_CHECK_IN	34134
Source Control/Undo Check Out...	IDC_SCC_UNDO_CHECK_OUT	34145
Source Control/Add to Source Control...	IDC_SCC_ADD	34133
Source Control/Remove from Source Control...	IDC_SCC_REMOVE	34143
Source Control/Show History...	IDC_SCC_HISTORY	34139
Source Control/Show Differences...	IDC_SCC_DIFF	34136
Source Control/Properties...	IDC_SCC_PROPERTIES	34141
Source Control/Refresh Status...	IDC_SCC_REFRESH	34142
Source Control/Run Native Interface...	IDC_SCC_RUN	34144
Add Files to Project...	IDC_PROJECT_ADD_FILES	34109
Add URL to Project...	IDC_PROJECT_ADD_URL	34112
Add Active File to Project	IDC_PROJECT_ADD_ACTIVE	34106
Add Active and Related Files to Project	IDC_PROJECT_ADD_RELATED	34111
Add Project Folder to Project...	IDC_PROJECT_ADD_FOLDER	34110
Add External Folder to Project...	IDC_PROJECT_ADD_EXT_FOLDER	34107
Add External Web Folder to Project...	IDC_PROJECT_ADD_EXT_URL_FOLDER	34108
Project Properties...	IDC_PROJECT_PROPERTIES	34124
Recent Project	IDC_RECENT_PROJECT_ID	34265

4.3.4 XML menu

Commands from the XML menu:

Menu Text	Command Name	ID
Insert/Attribute	IDC_INSERT_ATTRIBUTE	33449
Insert/Element	IDC_INSERT_STRUCT	33459
Insert/Text	IDC_INSERT_TEXT	33460
Insert/CData	IDC_INSERT_CDATA	33450
Insert/Comment	IDC_INSERT_COMMENT	33451
Insert/XML	IDC_INSERT_XML	33461
Insert/Processing Instruction	IDC_INSERT_PI	33458
Insert/XInclude	IDC_INSERT_XINCLUDE	34019
Insert/DOCTYPE	IDC_INSERT_DEF_DOCTYPE	33453
Insert/ExternalID	IDC_INSERT_DEF_EXTERNAL_ID	33456
Insert/ELEMENT	IDC_INSERT_DEF_ELEMENT	33454
Insert/ATTLIST	IDC_INSERT_DEF_ATTLIST	33452
Insert/ENTITY	IDC_INSERT_DEF_ENTITY	33455
Insert/NOTATION	IDC_INSERT_DEF_NOTATION	33457
Append/Attribute	IDC_APPEND_ATTRIBUTE	33415

Append/Element	IDC_APPEND_STRUCT	33425
Append/Text	IDC_APPEND_TEXT	33426
Append/CData	IDC_APPEND_CDATA	33416
Append/Comment	IDC_APPEND_COMMENT	33417
Append/XML	IDC_APPEND_XML	33427
Append/Processing Instruction	IDC_APPEND_PI	33424
Append/XInclude	IDC_APPEND_XINCLUDE	34026
Append/DOCTYPE	IDC_APPEND_DEF_DOCTYPE	33419
Append/ExternalID	IDC_APPEND_DEF_EXTERNAL_ID	33422
Append/ELEMENT	IDC_APPEND_DEF_ELEMENT	33420
Append/ATTLIST	IDC_APPEND_DEF_ATTLIST	33418
Append/ENTITY	IDC_APPEND_DEF_ENTITY	33421
Append/NOTATION	IDC_APPEND_DEF_NOTATION	33423
Add child/Attribute	IDC_ADD_CHILD_ATTRIBUTE	33402
Add child/Element	IDC_ADD_CHILD_STRUCT	33412
Add child/Text	IDC_ADD_CHILD_TEXT	33413
Add child/CData	IDC_ADD_CHILD_CDATA	33403
Add child/Comment	IDC_ADD_CHILD_COMMENT	33404
Add child/XML	IDC_ADD_CHILD_XML	33414
Add child/Processing Instruction	IDC_ADD_CHILD_PI	33411
Add child/XInclude	IDC_ADD_CHILD_XINCLUDE	34027
Add child/DOCTYPE	IDC_ADD_CHILD_DEF_DOCTYPE	33406
Add child/ExternalID	IDC_ADD_CHILD_DEF_EXTERNAL_ID	33409
Add child/ELEMENT	IDC_ADD_CHILD_DEF_ELEMENT	33407
Add child/ATTLIST	IDC_ADD_CHILD_DEF_ATTLIST	33405
Add child/ENTITY	IDC_ADD_CHILD_DEF_ENTITY	33408
Add child/NOTATION	IDC_ADD_CHILD_DEF_NOTATION	33410
Convert to/Attribute	IDC_CONVERT_TO_ATTRIBUTE	33429
Convert to/Element	IDC_CONVERT_TO_STRUCT	33439
Convert to/Text	IDC_CONVERT_TO_TEXT	33440
Convert to/CData	IDC_CONVERT_TO_CDATA	33430
Convert to/Comment	IDC_CONVERT_TO_COMMENT	33431
Convert to/XML	IDC_CONVERT_TO_XML	33441
Convert to/Processing Instruction	IDC_CONVERT_TO_PI	33438
Convert to/DOCTYPE	IDC_CONVERT_TO_DEF_DOCTYPE	33433
Convert to/ExternalID	IDC_CONVERT_TO_DEF_EXTERNAL_ID	33436
Convert to/ELEMENT	IDC_CONVERT_TO_DEF_ELEMENT	33434
Convert to/ATTLIST	IDC_CONVERT_TO_DEF_ATTLIST	33432
Convert to/ENTITY	IDC_CONVERT_TO_DEF_ENTITY	33435
Convert to/NOTATION	IDC_CONVERT_TO_DEF_NOTATION	33437

Table/Display as Table	IDC_GRID_VIEW_AS_TABLE	34075
Table/Insert Row	IDC_TABLE_INSERT_ROW	34158
Table/Append Row	IDC_TABLE_APPEND_ROW	34157
Table/Ascending Sort	IDC_TABLE_SORT_ASC	33464
Table/Descending Sort	IDC_TABLE_SORT_DESC	33465
Move left	IDC_MOVE_LEFT	34091
Move right	IDC_MOVE_RIGHT	34092
Enclose in Element	IDC_ENCLOSE_IN_ELEMENT	33446
Evaluate XPath...	IDC_EVALUATE_XPATH	34007
Check well-formedness	IDC_CHECK_WELL_FORM	34049
Validate	IDC_VALIDATE	34174
Update Entry Helpers	IDC_UPDATE_ELEMENT_CHOICE	34173
Namespace prefix...	IDC_NAMESPACE	33462

4.3.5 DTD/Schema Menu

Commands from the DTD/Schema menu:

Menu Text	Command Name	ID
Assign DTD...	IDC_ASSIGN_DTD	34032
Assign Schema...	IDC_ASSIGN_SCHEMA	34033
Include another DTD...	IDC_INCLUDE_DTD	34084
Go to DTD	IDC_GOTO_DTD	34072
Go to Schema	IDC_GOTO_SCHEMA	34074
Go to Definition	IDC_GOTO_DEFINITION	33447
Generate DTD/Schema...	IDC_GENERATE_DTD_SCHEMA	34068
Convert DTD/Schema...	IDC_CONVERT_DTD_SCHEMA	34052
Convert to UML...	IDC_CONVERT_SCHEMA_TO_UML	34008
Generate XML from DB, Excel, EDI with MapForce	IDC_DTD_OPENIN_MAPFORCE	34056
Design HTML/PDF Output in StyleVision...	IDC_DTD_OPENIN_STYLEVISION	34057
Generate Sample XML File...	IDC_GENERATE_XML_FROM_SCHEMA	34069
Generate Program Code...	IDC_GENERATE_CODE_FROM_SCHEMA	34067
Flush memory cache	IDC_FLUSH_CACHED_FILES	34066

4.3.6 Schema Design Menu

Commands from the Schema Design menu:

Menu Text	Command Name	ID
Schema Settings	IDC_SCHEMA_NAMESPACES	3357 1
Save Diagram...	IDC_SCHEMA_SAVE_DIAGRAM	3358 1

Generate Documentation	IDC_SCHEMA_DOCUMENTATION	3414 6
Configure view	IDC_SCHEMA_VIEW_CONFIG	3359 3
Zoom	IDC_SCHEMA_ZOOM	3415 0
Display All Globals	IDC_SCHEMA_MODE_GLOBALS	3414 7
Display Diagram	IDC_SCHEMA_MODE_DIAGRAM	3357 0
Enable Oracle Schema Extensions	IDC_SCHEMA_ORACLE_EXTENSIONS	3357 7
Oracle Schema Settings...	IDC_SCHEMA_ORACLE_SCHEMA_SETTING S	3357 8
Enable Microsoft SQL Server Schema Extensions	IDC_SCHEMA_SQLSERVER_EXTENSIONS	3358 8
Named Schema Relationships...	IDC_SCHEMA_SQLSERVER_GLOBAL_RELA TIONSHPIS	3358 9
Unnamed Element Relationships...	IDC_SCHEMA_SQLSERVER_LOCAL_RELATI ONSHIPS	3359 0
Connect to SchemaAgent Server...	IDC_SCHEMA_SCHEMAAGENT_SERVER_C ONNECT	3358 2
Disconnect from SchemaAgent Server	IDC_SCHEMA_SCHEMAAGENT_SERVER_DI SCONNECT	3358 3
Show in SchemaAgent/Schema only	IDC_SCHEMAAGENT_SHOW	3350 4
Show in SchemaAgent/Schema and all directly referenced	IDC_SCHEMAAGENT_SHOW_RELATED	3350 7
Show in SchemaAgent/Schema and all referenced	IDC_SCHEMAAGENT_SHOW_ALL	3350 5
Show in SchemaAgent/Schema and all linked	IDC_SCHEMAAGENT_SHOW_ALL_REACHA BLE	3350 6
Extended Validation	IDC_SCHEMA_EXTVALID_MENU	3353 9

4.3.7 XSL/XQuery Menu

Commands from the XSL/XQuery menu:

Menu Text	Command Name	ID
XSL Transformation	IDC_TRANSFORM_XSL	3300 6
XSL:FO Transformation	IDC_TRANSFORM_XSLFO	3300 7
XSL Parameters/XQuery Variables...	IDC_TRANSFORM_XSL_PARAMS	3300 8
XQuery Execution	IDC_TRANSFORM_XQUERY	3417 0
Enable XSLT 2 / XQuery profiling....	IDC_PROFILING_OPTIONS	3410 5
Assign XSL...	IDC_ASSIGN_XSL	3300 1
Assign XSL:FO...	IDC_ASSIGN_XSLFO	3300 2

Assign sample XML file...	IDC_ASSIGN_SAMPLE_XML	3300 0
Go to XSL	IDC_GOTO_XSL	3300 4
Start Debugger/Go	ID_PROCESS_XSL	3421 2
Stop Debugger	ID_XSLT_DEBUGGER_STOP	3301 7
Restart Debugger	ID_XSLT_DEBUGGER_RESTART	3301 3
End Debugger Session	ID_XSLT_DEBUGGER_END_SESSION	3301 1
Step Into	ID_XSLT_DEBUGGER_STEP	3301 4
Step Out	ID_XSLT_DEBUGGER_STEP_OUT	3301 5
Step Over	ID_XSLT_DEBUGGER_STEP_OVER	3301 6
Show Current Execution Node	ID_XSLT_DEBUGGER_GO_TO_CURRENT_EXECUTION_NODES	3301 2
Insert/Remove Breakpoint	IDC_TOGGLE_BREAKPOINT	3424 6
Insert/Remove Tracepoint	IDC_TOGGLE_TRACEPOINT	3424 8
Enable/Disable Breakpoint	IDC_ENABLE_BREAKPOINT	3424 5
Enable/Disable Tracepoint	IDC_ENABLE_TRACEPOINT	3424 7
Breakpoints/Tracepoints...	ID_XSLTDEBUGGER_BREAKPOINTS	3300 9
Debug Windows/Call Stack	ID_XSL_DEBUGWINDOWS_CALLSTACK	3423 8
Debug Windows/XPath-Watch	ID_XSL_DEBUGWINDOWS_WATCH	3424 4
Debug Windows/Context	ID_XSL_DEBUGWINDOWS_CONTEXT	3423 9
Debug Windows/Variables	ID_XSL_DEBUGWINDOWS_VARIABLE	3424 3
Debug Windows/Messages	ID_XSL_DEBUGWINDOWS_MESSAGES	3424 0
Debug Windows/Templates	ID_XSL_DEBUGWINDOWS_TEMPLATES	3424 1
Debug Windows/Info	ID_XSLXQUERY_DEBUGWINDOWS_INFO	3423 7
Debug Windows/Trace	ID_XSL_DEBUGWINDOWS_TRACES	3424 2
XSLT/XQuery Settings...	ID_XSLTDEBUGGER_SETTINGS	3301 0

4.3.8 Authentic Menu

Commands from the Authentic menu:

Menu Text	Command Name	ID

New Document...	IDC_AUTHENTIC_NEW_FILE	34036
Edit Database Data...	IDC_AUTHENTIC_EDIT_DB	34035
Assign a StyleVision Stylesheet...	IDC_ASSIGN_SPS	34034
Edit StyleVision Stylesheet	IDC_EDIT_SPS	34060
Define XML Entities...	IDC_DEFINE_ENTITIES	32805
Hide markup	IDC_MARKUP_HIDE	34087
Show Small markup	IDC_MARKUP_SMALL	34090
Show Large markup	IDC_MARKUP_LARGE	34088
Show Mixed markup	IDC_MARKUP_MIX	34089
Append row	IDC_ROW_APPEND	32806
Insert row	IDC_ROW_INSERT	32809
Duplicate row	IDC_ROW_DUPLICATE	32808
Move row Up	IDC_ROW_MOVE_UP	32811
Move row Down	IDC_ROW_MOVE_DOWN	32810
Delete row	IDC_ROW_DELETE	32807

4.3.9 Convert Menu

Commands from the Convert menu:

Menu Text	Command Name	ID
Import Text file...	IDC_IMPORT_TEXT	34082
Import Database data...	IDC_IMPORT_DATABASE	34080
Import Microsoft Word document...	IDC_IMPORT_WORD	34083
Create XML Schema from DB Structure	IDC_CREATE_DB_SCHEMA	34054
DB Import based on XML Schema	IDC_IMPORT_DB_SCHEMA	34081
Create DB Structure from XML Schema	IDC_CREATE_DB_BASED_ON_SCHEMA	34053
Export to Text files...	IDC_EXPORT_TEXTFILE	34064
Export to a Database...	IDC_EXPORT_DB	34003
Oracle XML DB/Search...	ID_CONVERT_ORACLEXMLDB_QUERY	34207
Oracle XML DB/List Schemas...	ID_ORACLEXMLDB_LISTSCHEMAS	34211
Oracle XML DB/Add Schema...	ID_CONVERT_ORACLEXMLDB_ADDSCHEMA	34204
Oracle XML DB/Browse Oracle XML Documents...	ID_CONVERT_ORACLEXMLDB_BROWSE	34205
Oracle XML DB/Properties...	ID_CONVERT_ORACLEXMLDB_PROPERTIES	34206

4.3.10 View Menu

Commands from the View menu:

Menu Text	Command Name	ID
Text view	IDC_VIEW_TEXT	34180
Enhanced Grid view	IDC_VIEW_GRID	34178

Schema/WSDL Design view	IDC_VIEW_SCHEMA	34179
Authentic view	IDC_VIEW_CONTENT	34177
Browser view	IDC_VIEW_BROWSER	34176
Expand +	IDC_SEL_EXPAND	34152
Collapse -	IDC_SEL_COLLAPSE	34151
Expand fully	IDC_SEL_EXPAND_ALL	33463
Collapse unselected	IDC_COLLAPSE_UNSELECTED	33428
Optimal widths	IDC_OPTIMAL_WIDTHS	34099
Word Wrap	IDC_WORD_WRAP	34181
Go to line/char	IDC_GOTO_LINE	34073
Go to File	IDC_GOTO_FILE	33448
Line Numbers Margin	IDC_TOGGLE_NUMLINEMARGIN	34168
Bookmarks Margin	IDC_TOGGLE_BOOKMARKMARGIN	34163
Source Folding Margin	IDC_TOGGLE_FOLDINGMARGIN	34166
Indentation Guides	IDC_TOGGLE_INDENTGUIDES	34167

4.3.11 Browser Menu

Commands from the Browser menu:

Menu Text	Command Name	ID
Back	IDC_BROWSER_BACK	34039
Forward	IDC_BROWSER_FORWARD	34045
Stop	IDC_BROWSER_STOP	34047
Refresh	IDC_BROWSER_REFRESH	34046
Fonts/Largest	IDC_BROWSER_FONT_LARGEST	34041
Fonts/Larger	IDC_BROWSER_FONT_LARGE	34040
Fonts/Medium	IDC_BROWSER_FONT_MEDIUM	34042
Fonts/Smaller	IDC_BROWSER_FONT_SMALL	34043
Fonts/Smallest	IDC_BROWSER_FONT_SMALLEST	34044
Separate window	IDC_BROWSER_USE_OWN_FRAME	34048

4.3.12 WSDL Menu

Commands from the WSDL menu:

Menu Text	Command Name	ID
Messages/Insert message	ID_WSDL_MESSAGES_ADDNEWMESSAGE	33715
Messages/Delete message	ID_WSDL_MESSAGES_DELETESELECTEDMESSAGE	33717
Messages/Add message part (parameter)	ID_WSDL_MESSAGES_ADDMESSAGEPART	33714
Messages/Delete message part (parameter)	ID_WSDL_MESSAGES_DELETEMESSAGEPART	33716

Operations/Insert Operation	ID_WSDL_OPERATIONS_INSERTOPERATION	33725
Operations/Delete operation	ID_WSDL_OPERATIONS_DELETEOPERATION	33724
Operations/Add input element	ID_WSDL_OPERATIONS_ADDINPUTFUNCTION	33719
Operations/Add output element	ID_WSDL_OPERATIONS_ADDOUTPUTFUNCTION	33721
Operations/Add fault element	ID_WSDL_OPERATIONS_ADDFAULTFUNCTION	33718
Operations/Delete input/output/fault element	ID_WSDL_OPERATIONS_DELETEINPUTOUTPUTFUNCTION	33723
Operations/Add new message to input/output/fault element	ID_WSDL_OPERATIONS_ADDNEWMESSEGETOTHISELEMENT	33720
Operations/empty operation	ID_WSDL_OPERATIONS_APPENDAOOPERATIONTOHISPORTTYPE	33722
PortType/Insert portType	ID_WSDL_PORTTYPE_INSERTAPORTTYPE	33727
PortType/Delete PortType	ID_WSDL_PORTTYPE_DELETETHISPORTTYPE	33726
Binding/Insert binding	ID_WSDL_BINDING_NEWBINDING	33713
Binding/Delete binding	ID_WSDL_BINDING_DELETEBINDING	33711
Binding/Append Child/soap:body	ID_WSDL_BINDING_APPENDEXTENSIBILITY_SOAPBODY	33706
Binding/Append Child/soap:header	ID_WSDL_BINDING_APPENDEXTENSIBILITY_SOAPHEADER	33708
Binding/Append Child/soap:headerfault	ID_WSDL_BINDING_APPENDEXTENSIBILITY_SOAPHEADERFAULT	33709
Binding/Append Child/soap:fault	ID_WSDL_BINDING_APPENDEXTENSIBILITY_SOAPFAULT	33707
Binding/Append Child/mime:content	ID_WSDL_BINDING_APPENDEXTENSIBILITY_MIMECONTENT	33702
Binding/Append Child/mime:multipartrelated	ID_WSDL_BINDING_APPENDEXTENSIBILITY_MIMEMULTIPARTRELATED	33704
Binding/Append Child/mime:part	ID_WSDL_BINDING_APPENDEXTENSIBILITY_MIMEPART	33705
Binding/Append Child/mime:mimeXml	ID_WSDL_BINDING_APPENDEXTENSIBILITY_MIMEMIMEXML	33703
Binding/Append Child/http:urlencoded	ID_WSDL_BINDING_APPENDEXTENSIBILITY_HTTPURLENCODED	33700
Binding/Append Child/http:urlreplacement	ID_WSDL_BINDING_APPENDEXTENSIBILITY_HTTPURLREPLACEMENT	33701
Binding/Delete extensibility element	ID_WSDL_BINDING_DELETEEXTESIBILITY	33712
Service/Insert service	ID_WSDL_SERVICE_INSERTSERVICE	33731
Service/Delete service	ID_WSDL_SERVICE_DELETETHISSERVICE	33729
Service/Insert port	ID_WSDL_SERVICE_INSERTNEWPORT	33730
Service/Delete port	ID_WSDL_SERVICE_DELETETHISPORT	33728

Types/New schema	ID_WSDL_TYPES_NEWSHEMA	3373 3
Types/Edit schema(s) in Schema View	ID_WSDL_TYPES_EDITTHISSHEMA	3373 2
Save Diagram...	IDC_WSDL_SAVE_DIAGRAM	3400 1
Generate Documentation...	ID_WSDL_GENERATEDOCUMENTATION	3423 3
Reparsing WSDL document	IDC_WSDL_REPARSE	3377 4

4.3.13 SOAPMenu

Commands from the SOAP menu:

Menu Text	Command Name	ID
Create new SOAP request	ID_SOAP_GENERATESOAPMESSAGE	34224
Send request to server	ID_SOAP_SENDREQUESTTOSERVER	34225
Change SOAP request parameters	ID_SOAP_SOAPREQUESTSETTINGS	34227
Soap Debugger Session	ID_SOAP_SOAPDEBUGGER	34226
Go	ID_SOAPDEBUGGER_BUTTONPLAY	34221
Single Step	ID_SOAPDEBUGGER_SINGLESTEP	34222
Break on next Request	ID_SOAPDEBUGGER_BREAKONNEXTREQUEST	34219
Break on next Response	ID_SOAPDEBUGGER_BREAKONNEXTRESPONSE	34220
Stop the proxy server	ID_SOAPDEBUGGER_STOPSERVER	34223
Soap Debugger Options	ID_SOAPDEBUGGEROPTIONS	34218

4.3.14 Tools Menu

Commands from the Tools menu:

Menu Text	Command Name	ID
Spelling...	IDC_SPELL_CHECK	34154
Spelling options...	IDC_SPELL_OPTIONS	34155
Switch to Scripting environment...	ID_WINDOW_SWITCHTOVBA	34231
Show macros...	ID_WINDOW_VBAMACROS	34232
Project/Assign Scripts to Project	ID_SCRIPTINGPRJ_ASSIGN	34214
Project/Unassign Scripts from Project	ID_SCRIPTINGPRJ_UNASSIGN	34215
Project/Project Scripts active	ID_SCRIPTINGPRJ_ACTIVE	34213
Compare open file with...	ID_XMLDIFF_CHOOSE_FILES	34235
Compare directories...	ID_XMLDIFF_CHOOSE_DIRECTORIES	34234
Compare options...	ID_XMLDIFF_SETTINGS	34236
Customize...	IDC_CUSTOMIZE	34055
Options...	IDC_SETTINGS	33300
<placeholder>	ID_SCRIPTING_MACROITEMS	34249

4.3.15 Window Menu

Commands from the Window menu:

Menu Text	Command Name	ID
Cascade	ID_WINDOW_CASCADE	57650
Tile horizontally	ID_WINDOW_TILE_HORZ	57651
Tile vertically	ID_WINDOW_TILE_VERT	57652
Project window	IDC_PROJECT_WINDOW	34128
Info window	IDC_INFO_WINDOW	34085
Entry Helpers	IDC_ENTRY_HELPERS	34062
Output windows	IDC_OUTPUT_DIALOGBARS	34004
Project and Entry Helpers	IDC_PROJECT_ENTRYHELPERS	34006
All on/off	IDC_ALL_BARS	34031

4.3.16 Help Menu

Commands from the Help menu:

Menu Text	Command Name	ID
Table of Contents...	IDC_HELP_CONTENTS	34076
Index...	IDC_HELP_INDEX	34077
Search...	IDC_HELP_SEARCH	34079
Keyboard Map...	IDC_HELP_KEYMAPDLG	34078
Software Activation...	IDC_ACTIVATION	34005
Order Form...	IDC_OPEN_ORDER_PAGE	34094
Registration...	IDC_REGISTRATION	34131
Check for Updates...	IDC_CHECK_FOR_UPDATES	34275
Support Center...	IDC_OPEN_SUPPORT_PAGE	34096
FAQ on the Web...	IDC_SHOW_FAQ	34153
Download Components and Free Tools...	IDC_OPEN_COMPONENTS_PAGE	34093
XMLSpy on the Internet..	IDC_OPEN_XML_SPY_HOME	34098
XMLSpy Training...	ID_HELP_XMLSPYTRAINING	34210
About XMLSpy...	ID_APP_ABOUT	57664

4.4 Accessing XMLSpyAPI

The focus of this documentation is the ActiveX controls and interfaces required to integrate the XMLSpy user interface into your application. To allow you to automate or control the functionality of the integrated components, the following properties give you access to the XMLSpy automation interface (XMLSpyAPI):

[XMLSpyControl.Application](#)
[XMLSpyControlDocument.Document](#)
[XMLSpyControlPlaceholder.Project](#)

Some restrictions apply to the usage of the XMLSpy automation interface when integrating XMLSpyControl at document-level. See [Integration at document level](#) for details.

4.5 Object Reference

Objects:

[XMLSpyCommand](#)
[XMLSpyCommands](#)
[XMLSpyControl](#)
[XMLSpyControlDocument](#)
[XMLSpyControlPlaceHolder](#)

To give access to standard XMLSpy functionality, objects of the **XMLSpy automation interface** can be accessed as well. See [XMLSpyControl.Application](#), [XMLSpyControlDocument.Document](#) and [XMLSpyControlPlaceHolder.Project](#) for more information.

4.5.1 XMLSpyCommand

Properties:

[ID](#)
[Label](#)
[IsSeparator](#)
[ToolTip](#)
[StatusText](#)
[Accelerator](#)
[SubCommands](#)

Description:

Each `Command` object can be one of three possible types:

- **Command:** `ID` is set to a value greater 0 and `Label` is set to the command name. `IsSeparator` is false and the `SubCommands` collection is empty.
- **Separator:** `IsSeparator` is true. `ID` is 0 and `Label` is not set. The `SubCommands` collection is empty.
- **(Sub) Menu:** The `SubCommands` collection contains [Command](#) objects and `Label` is the name of the menu. `ID` is set to 0 and `IsSeparator` is false.

Accelerator

Property: `Label` as [string](#)

Description:

For command objects that are children of the `ALL_COMMANDS` collection, this is the accelerator key defined for the command. If the command has no accelerator key assigned, this property returns the empty string.

The string representation of the accelerator key has the following format:

[ALT+] [CTRL+] [SHIFT+] key

Where `key` is converted using the Windows Platform SDK function `GetKeyNameText`.

ID

Property: `ID` as [long](#)

Description:

ID is 0 for separators and menus.

For commands, this is the ID which can be used with [Exec](#) and [QueryStatus](#).

IsSeparator

Property: `IsSeparator` as [boolean](#)

Description:

True if the command is a separator.

Label

Property: `Label` as [string](#)

Description:

Label is empty for separators.

For command objects that are children of the ALL_COMMANDS collection, this is a unique name. Command icons are stored in files with this name. See [Query Commands](#) for more information.

For command objects that are children of menus, the label property holds the command's menu text.

For sub-menus, this property holds the menu text.

StatusText

Property: `StatusText` as [string](#)

Description:

For command objects that are children of the ALL_COMMANDS collection, this is the text shown in the status bar when the command is selected.

SubCommands

Property: `SubCommands` as [Commands](#)

Description:

The `SubCommands` collection holds any sub-commands if this command is actually a menu or submenu.

ToolTip

Property: `ToolTip` as [string](#)

Description:

For command objects that are children of the ALL_COMMANDS collection, this is the text shown as tool-tip.

4.5.2 XMLSpyCommands

Properties:

[Count](#)

Item

Description:

Collection of [Command](#) objects to get access to command labels and IDs of the XMLSpyControl. Those commands can be executed with the [Exec](#) method and their status can be queried with [QueryStatus](#).

Count

Property: Count as [long](#)

Description:

Number of [Command](#) objects on this level of the collection.

Item

Property: Item (n as [long](#)) as [Command](#)

Description:

Gets the command with the index *n* in this collection. Index is 1-based.

4.5.3 XMLSpyControl

Properties:

[IntegrationLevel](#)

[Appearance](#)

[Application](#)

[BorderStyle](#)

[CommandsList](#)

CommandsStructure (deprecated)

[EnableUserPrompts](#)

[MainMenu](#)

[Toolbars](#)

Methods:

[Open](#)

[Exec](#)

[QueryStatus](#)

Events:

[OnUpdateCmdUI](#)

[OnOpenedOrFocused](#)

[OnCloseEditingWindow](#)

[OnFileChangedAlert](#)

[OnContextChanged](#)

[OnDocumentOpened](#)

[OnValidationWindowUpdated](#)

This object is a complete ActiveX control and should only be visible if the XMLSpy library is used in the Application Level mode.

CLSID: a258bba2-3835-4c16-8590-72b44f52c471

ProgID: Altova.XMLSpyControl

Properties

The following properties are defined:

[IntegrationLevel](#)
[EnableUserPrompts](#)
[Appearance](#)
[BorderStyle](#)

Command related properties:

[CommandsList](#)
[MainMenu](#)
[Toolbars](#)

CommandsStructure (deprecated)

Access to XMLSpyAPI:

[Application](#)

Appearance

Property: Appearance as [short](#)

Dispatch Id: -520

Description:

A value not equal to 0 displays a client edge around the control. Default value is 0.

Application

Property: Application as [Application](#)

Dispatch Id: 1

Description:

The [Application](#) property gives access to the [Application](#) object of the complete XMLSpy automation server API. The property is read-only.

BorderStyle

Property: BorderStyle as [short](#)

Dispatch Id: -504

Description:

A value of 1 displays the control with a thin border. Default value is 0.

CommandsList

Property: CommandList as [Commands](#) (read-only)

Dispatch Id: 1004

Description:

This property returns a flat list of all commands defined available with XMLSpyControl. For more information see [C# Sample](#).

EnableUserPrompts

Property: EnableUserPrompts as [boolean](#)

Dispatch Id: 1006

Description:

Setting this property to *false*, disables user prompts in the control. The default value is *true*.

IntegrationLevel

Property: IntegrationLevel as [ICActiveXIntegrationLevel](#)

Dispatch Id: 1000

Description:

The `IntegrationLevel` property determines the operation mode of the control. See also [Integration at the application level](#) and [Integration at document level](#) for more information.

Note: It is important to set this property immediately after the creation of the `XMLSpyControl` object.

MainMenu

Property: MainMenu as [Command](#)(read-only)

Dispatch Id: 1003

Description:

This property gives access to the description of the XMLSpyControl main menu. For more information see [C# Sample](#).

Toolbars

Property: Toolbars as [Commands](#)(read-only)

Dispatch Id: 1005

Description:

This property returns a list of all toolbar descriptions that describe all toolbars available with XMLSpyControl. For more information see [C# Sample](#).

Methods

The following methods are defined:

[Open](#)

[Exec](#)

[QueryStatus](#)

Exec

Method: `Exec (nCmdID as long) as boolean`

Dispatch Id: 6

Description:

`Exec` calls the XMLSpy command with the ID `nCmdID`. If the command can be executed, the method returns `true`. See also `CommandsStructure` to get a list of all available commands and [QueryStatus](#) to retrieve the status of any command.

Open

Method: `Open (strFilePath as string) as boolean`

Dispatch Id: 5

Description:

The result of the method depends on the extension passed in the argument `strFilePath`. If the file extension is `.sps`, a new document is opened. If the file extension is `.svp`, the corresponding project is opened. If a different file extension is passed into the method, the control tries to load the file as a new component into the active document.

Do not use this method to load documents or projects when using the control in document-level integration mode. Instead, use [XMLSpyControlDocument.Open](#) and [XMLSpyControlPlaceholder.OpenProject](#).

QueryStatus

Method: `QueryStatus (nCmdID as long) as long`

Dispatch Id: 7

Description:

`QueryStatus` returns the enabled/disabled and checked/unchecked status of the command specified by `nCmdID`. The status is returned as a bit mask.

Bit	Value	Name	Meaning
0	1	Supported	Set if the command is supported.
1	2	Enabled	Set if the command is enabled (can be executed).
2	4	Checked	Set if the command is checked.

This means that if `QueryStatus` returns 0 the command ID is not recognized as a valid XMLSpy command. If `QueryStatus` returns a value of 1 or 5, the command is disabled.

Events

The XMLSpyControl ActiveX control provides the following connection point events:

[OnUpdateCmdUI](#)

[OnOpenedOrFocused](#)

[OnCloseEditingWindow](#)

[OnFileChangedAlert](#)

[OnContextChanged](#)

[OnDocumentOpened](#)

[OnValidationWindowUpdated](#)

OnCloseEditingWindow

Event: OnCloseEditingWindow (i_strFilePath as [String](#)) as [boolean](#)

Dispatch Id: 1002

Description:

This event is triggered when XMLSpy needs to close an already open document. As an answer to this event, clients should close the editor window associated with *i_strFilePath*. Returning *true* from this event indicates that the client has closed the document. Clients can return *false* if no specific handling is required and XMLSpyControl should try to close the editor and destroy the associated document control.

OnContextChanged

Event: OnContextChanged (i_strContextName as [String](#), i_bActive as [bool](#)) as [bool](#)

Dispatch Id: 1004

Description:

This event is triggered when XMLSpy activates or de-activates one of the following operational contexts:

- XSLT Profiling - "XSLTProfiling" is passed as the context name
- XSLT / XQuery debugging - "DebuggingXSLT" is passed as the context name
- SOAP debugging - "DebuggingSOAP" is passed as the context name

OnDocumentOpened

Event: OnDocumentOpened (objDocument as [Document](#))

Dispatch Id: 1

Description:

This event is triggered whenever a document is opened. The argument *objDocument* is a *Document* object from the XMLSpy automation interface and can be used to query for more details about the document, or perform additional operations. When integrating on document-level, it is often better to use the event

[XMLSpyControl.Document.OnDocumentOpened](#) instead.

OnFileChangedAlert

Event: OnFileChangedAlert (i_strFilePath as String) as bool

Dispatch Id: 1001

Description:

This event is triggered when a file loaded with XMLSpyControl, is changed on the harddisk by another application. Clients should return true, if they handled the event, or false, if XMLSpy should handle it in its customary way, i.e. prompting the user for reload.

OnLicenseProblem

Event: OnLicenseProblem (i_strLicenseProblemText as String)

Dispatch Id: 1005

Description:

This event is triggered when XMLSpyControl detects that no valid license is available for this control. In case of restricted user licenses this can happen some time after the control has been initialized. Integrators should use this event to disable access to this control's functionality. After returning from this event, the control will block access to its functionality (e.g. show empty windows in its controls and return errors on requests).

OnOpenedOrFocused

Event: OnOpenedOrFocused (i_strFilePath as String, i_bOpenWithThisControl as bool)

Dispatch Id: 1000

Description:

When integrating at application level, this event informs clients that a document has been opened, or made active by XMLSpy.

When integrating at document level, this event instructs the client to open the file i_strFilePath in a document window. If the file is already open, the corresponding document window should be made the active window.

if i_bOpenWithThisControl is true, the document must be opened with XMLSpyControl, since internal access is required. Otherwise, the file can be opened with different editors.

OnToolWindowUpdated

Event: OnToolWindowUpdated(pToolWnd as long)

Dispatch Id: 1006

Description:

This event is triggered when the tool window is updated.

OnUpdateCmdUI**Event:** OnUpdateCmdUI ()**Dispatch Id:** 1003**Description:**

Called frequently to give integrators a good opportunity to check status of XMLSpy commands using [XMLSpyControl.QueryStatus](#). Do not perform long operations in this callback.

OnValidationWindowUpdated**Event:** OnValidationWindowUpdated ()**Dispatch Id:** 3**Description:**

This event is triggered whenever the validation output window, is updated with new information.

4.5.4 XMLSpyControlDocument

Properties:

[Appearance](#)
[BorderStyle](#)
[Document](#)
[IsModified](#)
[Path](#)
[ReadOnly](#)

Methods:

[Exec](#)
[New](#)
[Open](#)
[QueryStatus](#)
[Reload](#)
[Save](#)
[SaveAs](#)

Events:

[OnDocumentOpened](#)
[OnDocumentClosed](#)
[OnModifiedFlagChanged](#)
[OnContextChanged](#)
[OnFileChangedAlert](#)
[OnActivate](#)

If the XMLSpyControl is integrated in the Document Level mode each document is displayed in an own object of type XMLSpyControlDocument. The XMLSpyControlDocument contains only one document at the time but can be reused to display different files one after another.

This object is a complete ActiveX control.

CLSID: 52A552E6-2AB8-4e3e-B545-BE998233DDA0
ProgID: Altova.XMLSpyControlDocument

Properties

The following properties are defined:

[ReadOnly](#)
[IsModified](#)
[Path](#)
[Appearance](#)
[BorderStyle](#)

Access to XMLSpyAPI:

[Document](#)

Appearance

Property: Appearance as [short](#)

Dispatch Id: -520

Description:

A value not equal to 0 displays a client edge around the document control. Default value is 0.

BorderStyle

Property: BorderStyle as [short](#)

Dispatch Id: -504

Description:

A value of 1 displays the control with a thin border. Default value is 0.

Document

Property: Document as Document

Dispatch Id: 1

Description:

The `Document` property gives access to the `Document` object of the XMLSpy automation server API. This interface provides additional functionalities which can be used with the document loaded in the control. The property is read-only.

IsModified

Property: IsModified as [boolean](#) (read-only)

Dispatch Id: 1006

Description:

`IsModified` is *true* if the document content has changed since the last open, reload or save operation. It is *false*, otherwise.

Path

Property: Path as [string](#)

Dispatch Id: 1005

Description:

Sets or gets the full path name of the document loaded into the control.

ReadOnly

Property: ReadOnly as [boolean](#)

Dispatch Id: 1007

Description:

Using this property you can turn on and off the read-only mode of the document. If `ReadOnly` is `true` it is not possible to do any modifications.

Methods

The following methods are defined:

Document handling:

[New](#)

[Open](#)

[Reload](#)

[Save](#)

[SaveAs](#)

Command Handling:

[Exec](#)

[QueryStatus](#)

Exec

Method: Exec (`nCmdID` as [long](#)) as [boolean](#)

Dispatch Id: 8

Description:

`Exec` calls the XMLSpy command with the ID `nCmdID`. If the command can be executed, the method returns `true`. The client should call the `Exec` method of the document control if there is currently an active document available in the application.

See also `CommandsStructure` to get a list of all available commands and [QueryStatus](#) to retrieve the status of any command.

New

Method: New () as [boolean](#)

Dispatch Id: 1000**Description:**

This method initializes a new document inside the control..

Open

Method: Open (strFileName as string) as boolean

Dispatch Id: 1001**Description:**

Open loads the file strFileName as the new document into the control.

QueryStatus

Method: QueryStatus (nCmdID as long) as long

Dispatch Id: 9**Description:**

QueryStatus returns the enabled/disabled and checked/unchecked status of the command specified by nCmdID. The status is returned as a bit mask.

Bit	Value	Name	Meaning
0	1	Supported	Set if the command is supported.
1	2	Enabled	Set if the command is enabled (can be executed).
2	4	Checked	Set if the command is checked.

This means that if QueryStatus returns 0 the command ID is not recognized as a valid XMLSpy command. If QueryStatus returns a value of 1 or 5 the command is disabled. The client should call the QueryStatus method of the document control if there is currently an active document available in the application.

Reload

Method: Reload () as boolean

Dispatch Id: 1002**Description:**

Reload updates the document content from the file system.

Save

Method: Save () as boolean

Dispatch Id: 1003

Description:

Save saves the current document at the location [Path](#).

SaveAs

Method: SaveAs (strFileName as [string](#)) as [boolean](#)

Dispatch Id: 1004

Description:

SaveAs sets [Path](#) to *strFileName* and then saves the document to this location.

Events

The XMLSpyControlDocument ActiveX control provides following connection point events:

[OnDocumentOpened](#)

[OnDocumentClosed](#)

[OnModifiedFlagChanged](#)

[OnContextChanged](#)

[OnFileChangedAlert](#)

[OnActivate](#)

[OnSetEditorTitle](#)

OnActivate

Event: OnActivate ()

Dispatch Id: 1005

Description:

This event is triggered when the document control is activated, has the focus, and is ready for user input.

OnContextChanged

Event: OnContextChanged (i_strContextName as [String](#), i_bActive as [bool](#)) as [bool](#)

Dispatch Id: 1004

Description:

This event is triggered when this document is shown in a different XMLSpy view. The following values are passed:

- Grid view - "View_0" is passed as the context name
- Text view - "View_1" is passed as the context name
- Browser view - "View_2" is passed as the context name
- Schema view - "View_3" is passed as the context name
- Authentic view - "View_4" is passed as the context name
- WSDL view - "View_5" is passed as the context name

OnDocumentClosed

Event: OnDocumentClosed (objDocument as Document)

Dispatch Id: 1001

Description:

This event is triggered whenever the document loaded into this control is closed. The argument objDocument is a Document object from the XMLSpy automation interface and should be used with care.

OnDocumentOpened

Event: OnDocumentOpened (objDocument as Document)

Dispatch Id: 1000

Description:

This event is triggered whenever a document is opened in this control. The argument objDocument is a Document object from the XMLSpy automation interface, and can be used to query for more details about the document, or perform additional operations.

OnDocumentSaveAs

Event: OnContextDocumentSaveAs (i_strFileName as String)

Dispatch Id: 1007

Description:

This event is triggered when this document gets internally saved under a new name.

OnFileChangedAlert

Event: OnFileChangedAlert () as bool

Dispatch Id: 1003

Description:

This event is triggered when the file loaded into this document control, is changed on the harddisk by another application. Clients should return true, if they handled the event, or false, if XMLSpy should handle it in its customary way, i.e. prompting the user for reload.

OnModifiedFlagChanged

Event: OnModifiedFlagChanged (i_bIsModified as boolean)

Dispatch Id: 1002

Description:

This event gets triggered whenever the document changes between modified and unmodified state. The parameter i_bIsModified is true if the document contents differs from the original content, and false, otherwise.

OnSetEditorTitle

Event: OnSetEditorTitle ()

Dispatch Id: 1006

Description:

This event is being raised when the contained document is being internally renamed.

4.5.5 XMLSpyControlPlaceholder

Properties available for all kinds of placeholder windows:

[PlaceholderWindowID](#)

Properties for project placeholder window:

[Project](#)

Methods for project placeholder window:

[OpenProject](#)

[CloseProject](#)

The XMLSpyControlPlaceholder control is used to show the additional XMLSpy windows like Overview, Library or Project window. It is used like any other ActiveX control and can be placed anywhere in the client application.

CLSID: 135DEEF4-6DF0-47c2-8F8C-F145F5F3F672

ProgID: Altova.XMLSpyControlPlaceholder

Properties

The following properties are defined:

[PlaceholderWindowID](#)

Access to XMLSpyAPI:

[Project](#)

Label

Property: Label as [String](#) (read-only)

Dispatch Id: 1001

Description:

This property gives access to the title of the placeholder. The property is read-only.

PlaceholderWindowID

Property: PlaceholderWindowID as [XMLSpyControlPlaceholderWindow](#)

Dispatch Id: 1

Description:

Using this property the object knows which XMLSpy window should be displayed in the client area of the control. The `PlaceholderWindowID` can be set at any time to any valid value of the [XMLSpyControlPlaceholderWindow](#) enumeration. The control changes its state immediately and shows the new XMLSpy window.

Project

Property: `Project` as `Project` (read-only)

Dispatch Id: 2

Description:

The `Project` property gives access to the `Project` object of the XMLSpy automation server API. This interface provides additional functionalities which can be used with the project loaded into the control. The property will return a valid project interface only if the placeholder window has [PlaceholderWindowID](#) with a value of `XMLSpyXProjectWindow ( =3 )`. The property is read-only.

Methods

The following method is defined:

[OpenProject](#)

[CloseProject](#)

OpenProject

Method: `OpenProject (strFileName as string) as boolean`

Dispatch Id: 3

Description:

`OpenProject` loads the file `strFileName` as the new project into the control. The method will fail if the placeholder window has a [PlaceholderWindowID](#) different to `XMLSpyXProjectWindow ( =3 )`.

CloseProject

Method: `CloseProject ()`

Dispatch Id: 4

Description:

`CloseProject` closes the project loaded the control. The method will fail if the placeholder window has a [PlaceholderWindowID](#) different to `XMLSpyXProjectWindow ( =3 )`.

Events

The XMLSpyControlPlaceholder ActiveX control provides following connection point events:

[OnModifiedFlagChanged](#)

OnModifiedFlagChanged

Event: OnModifiedFlagChanged (i_bIsModified as [boolean](#))

Dispatch Id: 1

Description:

This event gets triggered only for placeholder controls with a [PlaceholderWindowID](#) of XMLSpyXProjectWindow (=3). The event is fired whenever the project content changes between modified and unmodified state. The parameter *i_bIsModified* is *true* if the project contents differs from the original content, and *false*, otherwise.

OnSetLabel

Event: OnSetLabel(i_strNewLabel as [string](#))

Dispatch Id: 1000

Description:

Raised when the title of the placeholder window is changed.

4.5.6 Enumerations

The following enumerations are defined:

[IActiveXIntegrationLevel](#)
[XMLSpyControlPlaceholderWindow](#)

IActiveXIntegrationLevel

Possible values for the [IntegrationLevel](#) property of the XMLSpyControl.

```
IActiveXIntegrationOnApplicationLevel = 0
IActiveXIntegrationOnDocumentLevel   = 1
```

XMLSpyControlPlaceholderWindow

This enumeration contains the list of the supported additional XMLSpy windows.

```
XMLSpyControlNoToolWnd           = -1
XMLSpyControlEntryHelperTopToolWnd = 0
XMLSpyControlEntryHelperMiddleToolWnd = 1
XMLSpyControlEntryHelperBottomToolWnd = 2
XMLSpyControlValidatorOutputToolWnd = 3
XMLSpyControlProjectWindowToolWnd = 4
XMLSpyControlXSLTDebuggerContextToolWnd = 5
XMLSpyControlXSLTDebuggerCallstackToolWnd = 6
XMLSpyControlXSLTDebuggerVariableToolWnd = 7
XMLSpyControlXSLTDebuggerWatchToolWnd = 8
XMLSpyControlXSLTDebuggerTemplateToolWnd = 9
XMLSpyControlXSLTDebuggerInfoToolWnd = 10
XMLSpyControlXSLTDebuggerMessageToolWnd = 11
XMLSpyControlXSLTDebuggerTraceToolWnd = 12
XMLSpyControlSOAPDebuggerToolWnd = 13
XMLSpyControlXPathProfilerListToolWnd = 14
```

XMLSpyControlXPathProfilerTreeToolWnd	=	15
XMLSpyControlXPathDialogToolWnd	=	16
XMLSpyControlDBQueryManagerToolWnd	=	17
XMLSpyControlInfoToolWnd	=	18
XMLSpyControlXSLOutlineToolWnd	=	19
XMLSpyControlSchemaFindToolWnd	=	20

Altova XMLSpy 2012

Appendices

Appendices

These appendices contain technical information about XMLSpy and important licensing information. Each appendix contains sub-sections as given below:

Engine Information

- [Altova XSLT 1.0 Engine](#)
- [Altova XSLT 2.0 Engine](#)
- [Altova XQuery 1.0 Engine](#)
- [XPath 2.0 and XQuery 1.0 Functions](#)
- [Extensions](#)

Datatype Conversions between DBs and XML Schemas

- [DBs to XML Schemas](#)
- [XML Schemas to DBs](#)

Technical Data

- [OS and memory requirements](#)
- [Altova XML Parser](#)
- [Altova XSLT and XQuery Engines](#)
- [Unicode support](#)
- [Internet usage](#)

License Information

- [Electronic software distribution](#)
- [Software activation and license metering](#)
- [Copyrights](#)
- [End User License Agreement](#)

1 Engine Information

This section contains information about implementation-specific features of the [Altova XSLT 1.0 Engine](#), [Altova XSLT 2.0 Engine](#), and [Altova XQuery 1.0 Engine](#).

1.1 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</i>.</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</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.
```

1.2 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](#).

1.2.1 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"
  ...
>
```

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</i>.</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</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 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](#).

1.2.2 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`.

1.3 XQuery 1.0 Engine: Implementation Information

The Altova XQuery 1.0 Engine is built into Altova's XMLSpy and MapForce XML products. It is also available in the free AltovaXML package. This section provides information about implementation-defined aspects of behavior.

Standards conformance

The Altova XQuery 1.0 Engine conforms to the World Wide Web Consortium's (W3C's) [XQuery 1.0 Recommendation](#) of 23 January 2007. The XQuery standard gives implementations discretion about how to implement many features. Given below is a list explaining how the Altova XQuery 1.0 Engine implements these features.

Schema awareness

The Altova XQuery 1.0 Engine is **schema-aware**.

Encoding

The UTF-8 and UTF-16 character encodings are supported.

Namespaces

The following namespace URIs and their associated bindings are pre-defined.

Namespace Name	Prefix	Namespace URI
XML Schema types	xs:	http://www.w3.org/2001/XMLSchema
Schema instance	xsi:	http://www.w3.org/2001/XMLSchema-instance
Built-in functions	fn:	http://www.w3.org/2005/xpath-functions
Local functions	local:	http://www.w3.org/2005/xquery-local-functions

The following points should be noted:

- The Altova XQuery 1.0 Engine recognizes the prefixes listed above as being bound to the corresponding namespaces.
- Since the built-in functions namespace listed above is the default functions namespace in XQuery, the `fn:` prefix does not need to be used when built-in functions are invoked (for example, `string("Hello")` will call the `fn:string` function). However, the prefix `fn:` can be used to call a built-in function without having to declare the namespace in the query prolog (for example: `fn:string("Hello")`).
- You can change the default functions namespace by declaring the `default function namespace` expression in the query prolog.
- When using types from the XML Schema namespace, the prefix `xs:` may be used without having to explicitly declare the namespaces and bind these prefixes to them in the query prolog. (Example: `xs:date` and `xs:yearMonthDuration`.) If you wish to use some other prefix for the XML Schema namespace, this must be explicitly declared in the query prolog. (Example: `declare namespace alt = "http://www.w3.org/2001/XMLSchema"; alt:date("2004-10-04")`.)
- Note that the `untypedAtomic`, `dayTimeDuration`, and `yearMonthDuration` datatypes have been moved, with the CRs of 23 January 2007, from the XPath Datatypes namespace to the XML Schema namespace, so: `xs:yearMonthDuration`.

If namespaces for functions, type constructors, node tests, etc are wrongly assigned, an error is

reported. Note, however, that some functions have the same name as schema datatypes, e.g. `fn:string` and `fn:boolean`. (Both `xs:string` and `xs:boolean` are defined.) The namespace prefix determines whether the function or type constructor is used.

XML source document and validation

XML documents used in executing an XQuery document with the Altova XQuery 1.0 Engine must be well-formed. However, they do not need to be valid according to an XML Schema. If the file is not valid, the invalid file is loaded without schema information. If the XML file is associated with an external schema and is valid according to it, then post-schema validation information is generated for the XML data and will be used for query evaluation.

Static and dynamic type checking

The static analysis phase checks aspects of the query such as syntax, whether external references (e.g. for modules) exist, whether invoked functions and variables are defined, and so on. No type checking is done in the static analysis phase. If an error is detected in the static analysis phase, it is reported and the execution is stopped.

Dynamic type checking is carried out at run-time, when the query is actually executed. If a type is incompatible with the requirement of an operation, an error is reported. For example, the expression `xs:string("1") + 1` returns an error because the addition operation cannot be carried out on an operand of type `xs:string`.

Library Modules

Library modules store functions and variables so they can be reused. The Altova XQuery 1.0 Engine supports modules that are stored in a **single external XQuery file**. Such a module file must contain a `module` declaration in its prolog, which associates a target namespace. Here is an example module:

```
module namespace libns="urn:module-library";
declare variable $libns:company := "Altova";
declare function libns:webaddress() { "http://www.altova.com" };
```

All functions and variables declared in the module belong to the namespace associated with the module. The module is used by importing it into an XQuery file with the `import module` statement in the query prolog. The `import module` statement only imports functions and variables declared directly in the library module file. As follows:

```
import module namespace modlib = "urn:module-library" at
    "modulefilename.xq";
if ($modlib:company = "Altova")
then modlib:webaddress()
else error("No match found.")
```

External functions

External functions are not supported, i.e. in those expressions using the `external` keyword, as in:

```
declare function hoo($param as xs:integer) as xs:string external;
```

Collations

The default collation is the Unicode codepoint collation. No other collation is currently supported. Comparisons, including the `fn:max` function, are based on this collation.

Character normalization

No character normalization form is supported.

Precision of numeric types

- The `xs:integer` datatype is arbitrary-precision, i.e. it can represent any number of digits.
- The `xs:decimal` datatype has a limit of 20 digits after the decimal point.
- The `xs:float` and `xs:double` datatypes have limited-precision of 15 digits.

XQuery Instructions Support

The `Pragma` instruction is not supported. If encountered, it is ignored and the fallback expression is evaluated.

XQuery Functions Support

For information about implementation-specific behavior of XQuery 1.0 functions, see the section, [XPath 2.0 and XQuery 1.0 Functions](#).

1.4 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.

1.4.1 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.

1.4.2 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 <code>base-uri()</code> 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 <code>xml:base</code> 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: <pre> <collection> <doc href="uri-1" /> <doc href="uri-2" /> <doc href="uri-3" /> </collection> </pre> The files referenced by the <code>href</code> 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 <code>?</code> and <code>*</code> 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 <code>collection("c:\MyDocs*.xml")//Title</code> returns a sequence of all <code>DocTitle</code> elements in the <code>.xml</code> files in the <code>MyDocs</code> folder. • XQuery example: The expression <code>{for \$i in collection(c:\MyDocs*.xml) return element doc{base-uri(\$i)}}</code> returns the base URIs of all the <code>.xml</code> files in the <code>MyDocs</code> folder, each URI being within a <code>doc</code> element. • The default collection is empty.

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.

Function Name	Notes
position	See note on whitespace in the General Information section.
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.

1.5 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 and XQuery queries. 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.

1.5.1 Java Extension Functions

A Java extension function can be used within an XPath or XQuery expression to invoke a Java constructor or call a Java method (static or instance).

A field in a Java class is considered to be a method without any argument. A field can be static or instance. How to access fields is described in the respective sub-sections, static and instance.

This section is organized into the following sub-sections:

- [Java: Constructors](#)
- [Java: Static Methods and Static Fields](#)
- [Java: Instance Methods and Instance Fields](#)
- [Datatypes: 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.

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()" />
  </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
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/Carl.jar! /" >
<xsl:output exclude-result-prefixes="fn car xsl xs"/>

<xsl:template match="/">
  <xsl:variable name="myCar" select="car: Carl.new( ' red' ) " />
  <a><xsl:value-of select="car: Carl.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:

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.

<code>xs:string</code>	<code>java.lang.String</code>
<code>xs:boolean</code>	<code>boolean (primitive)</code> , <code>java.lang.Boolean</code>
<code>xs:integer</code>	<code>int</code> , <code>long</code> , <code>short</code> , <code>byte</code> , <code>float</code> , <code>double</code> , and the wrapper classes of these, such as <code>java.lang.Integer</code>
<code>xs:float</code>	<code>float (primitive)</code> , <code>java.lang.Float</code> , <code>double (primitive)</code>
<code>xs:double</code>	<code>double (primitive)</code> , <code>java.lang.Double</code>
<code>xs:decimal</code>	<code>float (primitive)</code> , <code>java.lang.Float</code> , <code>double (primitive)</code> , <code>java.lang.Double</code>

Subtypes of the XML Schema datatypes listed above (and which are used in XPath and XQuery) will also be converted to the Java type/s corresponding to that subtype's ancestor type.

In some cases, it might not be possible to select the correct Java method based on the supplied information. For example, consider the following case.

- The supplied argument is an `xs:untypedAtomic` value of 10 and it is intended for the method `mymethod(float)`.
- However, there is another method in the class which takes an argument of another datatype: `mymethod(double)`.
- Since the method names are the same and the supplied type (`xs:untypedAtomic`) could be converted correctly to either `float` or `double`, it is possible that `xs:untypedAtomic` is converted to `double` instead of `float`.
- Consequently the method selected will not be the required method and might not produce the expected result. To work around this, you can create a user-defined method with a different name and use this method.

Types that are not covered in the list above (for example `xs:date`) will not be converted and will generate an error. However, note that in some cases, it might be possible to create the required Java type by using a Java constructor.

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).

1.5.2 .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:

<code>asm</code>	The name of the assembly to be loaded.
<code>ver</code>	The version number (maximum of four integers separated by periods).
<code>sn</code>	The key token of the assembly's strong name (16 hex digits).
<code>from</code>	A URI that gives the location of the assembly (DLL) to be loaded. If the URI is relative, it is relative to the XSLT or XQuery document. If this parameter is present, any other parameter is ignored.
<code>partialname</code>	The partial name of the assembly. It is supplied to <code>Assembly.LoadWith.PartialName()</code> , which will attempt to load the assembly. If <code>partialname</code> is present, any other parameter is ignored.

`loc` The locale, for example, `en-US`. The default is `neutral`.

If the assembly is to be loaded from a DLL, use the `from` parameter and omit the `sn` parameter. If the assembly is to be loaded from the Global Assembly Cache (GAC), use the `sn` parameter and omit the `from` parameter.

A question mark must be inserted before the first parameter, and parameters must be separated by a semi-colon. The parameter name gives its value with an equals sign (see *example below*).

Examples of namespace declarations

An example of a namespace declaration in XSLT that identifies the system class

`System.Environment:`

```
xmlns:mys="clitype:System.Environment"
```

An example of a namespace declaration in XSLT that identifies the class to be loaded as

`Trade.Forward.Scrip:`

```
xmlns:mys="clitype:Trade.Forward.Scrip?asm=forward;version=10.6.2.1"
```

An example of a namespace declaration in XQuery that identifies the system class

`MyManagedDLL.testClass:`. Two cases are distinguished:

1. When the assembly is loaded from the GAC:

```
declare namespace cs="clitype:MyManagedDLL.testClass?asm=MyManagedDLL;
ver=1.2.3.4;loc=neutral;sn=b9f091b72dccfba8";
```

2. When the assembly is loaded from the DLL (complete and partial references below):

```
declare namespace
cs="clitype:MyManagedDLL.testClass?from=file:///C:/Altova
Projects/extFunctions/MyManagedDLL.dll;

declare namespace
cs="clitype:MyManagedDLL.testClass?from=MyManagedDLL.dll;
```

XSLT example

Here is a complete XSLT example that calls functions in system class `System.Math:`

```
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:fn="http://www.w3.org/2005/xpath-functions">
  <xsl:output method="xml" omit-xml-declaration="yes" />
  <xsl:template match="/">
    <math xmlns:math="clitype:System.Math">
      <sqrt><xsl:value-of select="math:Sqrt(9)"/></sqrt>
      <pi><xsl:value-of select="math:PI()"/></pi>
      <e><xsl:value-of select="math:E()"/></e>
      <pow><xsl:value-of select="math:Pow(math:PI(), math:E())"/></pow>
    </math>
  </xsl:template>
</xsl:stylesheet>
```

The namespace declaration on the element `math` associates the prefix `math:` with the URI `clitype:System.Math`. The `clitype:` beginning of the URI indicates that what follows identifies either a system class or a loaded class. The `math:` prefix in the XPath expressions associates the extension functions with the URI (and, by extension, the class) `System.Math`. The extension functions identify methods in the class `System.Math` and supply arguments

where required.

XQuery example

Here is an XQuery example fragment similar to the XSLT example above:

```
<math xmlns:math="clitype: System. Math">
  { math: Sqrt( 9) }
</math>
```

As with the XSLT example above, the namespace declaration identifies the .NET class, in this case a system class. The XQuery expression identifies the method to be called and supplies the argument.

.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

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.

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

Subtypes of the XML Schema datatypes listed above (and which are used in XPath and XQuery) will also be converted to the .NET type/s corresponding to that subtype's ancestor type.

In some cases, it might not be possible to select the correct .NET method based on the supplied information. For example, consider the following case.

- The supplied argument is an `xs: untypedAtomic` value of 10 and it is intended for the method `mymethod(float)`.
- However, there is another method in the class which takes an argument of another datatype: `mymethod(double)`.
- Since the method names are the same and the supplied type (`xs: untypedAtomic`) could be converted correctly to either `float` or `double`, it is possible that `xs: untypedAtomic` is converted to `double` instead of `float`.
- Consequently the method selected will not be the required method and might not produce the expected result. To work around this, you can create a user-defined method with a different name and use this method.

Types that are not covered in the list above (for example `xs: date`) will not be converted and will generate an error.

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).

1.5.3 MSXSL Scripts for XSLT

The `<msxsl:script>` element contains user-defined functions and variables that can be called from within XPath expressions in the XSLT stylesheet. The `<msxsl:script>` is a top-level element, that is, it must be a child element of `<xsl:stylesheet>` or `<xsl:transform>`.

The `<msxsl:script>` element must be in the namespace `urn:schemas-microsoft-com:xslt` (see *example below*).

Scripting language and namespace

The scripting language used within the block is specified in the `<msxsl:script>` element's `language` attribute and the namespace to be used for function calls from XPath expressions is identified with the `implements-prefix` attribute (see *below*).

```
<msxsl:script language="scripting-language" implements-prefix="user-namespace-prefix">
```

```
    function-1 or variable-1
    ...
    function-n or variable-n
```

```
</msxsl:script>
```

The `<msxsl:script>` element interacts with the Windows Scripting Runtime, so only languages that are installed on your machine may be used within the `<msxsl:script>` element. **The .NET**

Framework 2.0 platform or higher must be installed for MSXSL scripts to be used.

Consequently, the .NET scripting languages can be used within the `<msxsl:script>` element.

The `language` attribute accepts the same values as the `language` attribute on the HTML `<script>` element. If the `language` attribute is not specified, then Microsoft JScript is assumed as the default.

The `implements-prefix` attribute takes a value that is a prefix of a declared in-scope namespace. This namespace typically will be a user namespace that has been reserved for a function library. All functions and variables defined within the `<msxsl:script>` element will be in the namespace identified by the prefix specified in the `implements-prefix` attribute. When a function is called from within an XPath expression, the fully qualified function name must be in the same namespace as the function definition.

Example

Here is an example of a complete XSLT stylesheet that uses a function defined within a `<msxsl:script>` element.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:fn="http://www.w3.org/2005/xpath-functions"
  xmlns:msxsl="urn:schemas-microsoft-com:xslt"
  xmlns:user="http://mycompany.com/mynamespace">

  <msxsl:script language="VBScript" implements-prefix="user">
    <![CDATA[
      ' Input: A currency value: the wholesale price
      ' Returns: The retail price: the input value plus 20% margin,
      ' rounded to the nearest cent
      dim a as integer = 13
      Function AddMargin(WholesalePrice) as integer
        AddMargin = WholesalePrice * 1.2 + a
      End Function
    ]]>
  </msxsl:script>

  <xsl:template match="/">
    <html>
      <body>
        <p>
          <b>Total Retail Price =
            $<xsl:value-of select="user:AddMargin(50)" />
          </b>
          <br/>
          <b>Total Wholesale Price =
            $<xsl:value-of select="50" />
          </b>
        </p>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Datatypes

The values of parameters passed into and out of the script block are limited to XPath datatypes. This restriction does not apply to data passed among functions and variables within the script block.

Assemblies

An assembly can be imported into the script by using the `msxsl:assembly` element. The assembly is identified via a name or a URI. The assembly is imported when the stylesheet is compiled. Here is a simple representation of how the `msxsl:assembly` element is to be used.

```
<msxsl:script>
  <msxsl:assembly name="myAssembly.assemblyName" />
  <msxsl:assembly href="pathToAssembly" />
  ...
</msxsl:script>
```

The assembly name can be a full name, such as:

```
"system.Math, Version=3.1.4500.1 Culture=neutral
PublicKeyToken=a46b3f648229c514"
```

or a short name, such as `"myAssembly.Draw"`.

Namespaces

Namespaces can be declared with the `msxsl:using` element. This enables assembly classes to be written in the script without their namespaces, thus saving you some tedious typing. Here is how the `msxsl:using` element is used so as to declare namespaces.

```
<msxsl:script>
  <msxsl:using namespace="myAssemblyNS.NamespaceName" />
  ...
</msxsl:script>
```

The value of the `namespace` attribute is the name of the namespace.

1.5.4 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\(\)](#)

Chart functions (Enterprise and Reporting Editions only)

Altova extension functions for charts are supported only in the Enterprise and Reporting Editions of Altova products and enable charts to be generated from XML data. The chart functions have been organized into two groups:

- [Functions to generate and save charts](#)
- [Functions to create charts](#)

A third section gives a listing of the [chart data XML structure](#), from which charts can be generated. Finally, an [example XSLT document](#) shows how chart functions can be used to generate charts from XML data.

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.

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()

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.

Chart Functions

The chart functions listed below enable you to create, generate, and save charts as images. They are supported in the current version of your Altova product in the manner described below. However, note that in future versions of your product, support for one or more of these functions might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

The chart functions are organized into two groups:

- [Functions for generating and saving charts](#)
- [Functions for creating charts](#)

Note: Chart functions are supported only in the **Enterprise and Reporting Editions** of Altova products.

Functions for generating and saving charts

These functions take the chart object (obtained with the chart creation functions) and either generate an image or save an image to file

altova:generate-chart-image(\$chart, \$width, \$height, \$encoding) as atomic

where

- \$chart is the chart extension item obtained with the **altova:create-chart** function
- \$width and \$height must be specified with a length unit
- \$encoding may be `binarytobase64` or `binarytobase16`

The function returns the chart image in the specified encoding.

altova:generate-chart-image(\$chart, \$width, \$height, \$encoding, \$imagetype)
as atomic

where

- \$chart is the chart extension item obtained with the **altova:create-chart** function
- \$width and \$height must be specified with a length unit
- \$encoding may be `base64Binary` or `hexBinary`

- `$imagetype` may be one of the following image formats: `png, gif, bmp, jpg, jpeg`

The function returns the chart image in the specified encoding and image format.

altova: save-chart-image(`$chart`, `$filename`, `$width`, `$height`) as `empty()`

where

- `$chart` is the chart extension item obtained with the `altova: create-chart` function
- `$filename` is the path to and name of the file to which the chart image is to be saved
- `$width` and `$height` must be specified with a length unit

The function saves the chart image to the file specified in `$filename`.

altova: save-chart-image(`$chart`, `$filename`, `$width`, `$height`, `$imagetype`) as `empty()`

where

- `$chart` is the chart extension item obtained with the `altova: create-chart` function
- `$filename` is the path to and name of the file to which the chart image is to be saved
- `$width` and `$height` must be specified with a length unit
- `$imagetype` may be one of the following image formats: `png, gif, bmp, jpg, jpeg`

The function saves the chart image to the file specified in `$filename` in the image format specified.

Functions for creating charts

The following functions are used to create charts.

altova: create-chart(`$chart-config`, `$chart-data-series*`) as chart extension item

where

- `$chart-config` is the chart-config extension item obtained with the `altova: create-chart-config` function or via the `altova: create-chart-config-from-xml` function
- `$chart-data-series` is the chart-data-series extension item obtained with the `altova: create-chart-data-series` function or `altova: create-chart-data-series-from-rows` function

The function returns a chart extension item, which is created from the data supplied via the arguments.

altova: create-chart-config(`$type-name`, `$title`) as chart-config extension item

where

- `$type-name` specifies the type of chart to be created: `Pie`, `Pie3d`, `BarChart`, `BarChart3d`, `BarChart3dGrouped`, `LineChart`, `ValueLineChart`, `RoundGauge`, `BarGauge`
- `$title` is the name of the chart

The function returns a chart-config extension item containing the configuration information of the chart.

```
altova: create-chart-config-from-xml( $xml-struct ) as chart-config extension item
```

where

- `$xml-struct` is the XML structure containing the configuration information of the chart

The function returns a chart-config extension item containing the configuration information of the chart. This information is supplied in an [XML data fragment](#).

```
altova: create-chart-data-series( $series-name?, $x-values*, $y-values* ) as chart-data-series extension item
```

where

- `$series-name` specifies the name of the series
- `$x-values` gives the list of X-Axis values
- `$y-values` gives the list of Y-Axis values

The function returns a chart-data-series extension item containing the data for building the chart: that is, the names of the series and the Axes data.

```
altova: create-chart-data-row( x, y1, y2, y3, ... ) as chart-data-x-Ny-row extension item
```

where

- `x` is the value of the X-Axis column of the chart data row
- `yN` are the values of the Y-Axis columns

The function returns a chart-data-x-Ny-row extension item, which contains the data for the X-Axis column and Y-Axis columns of a single series.

```
altova: create-chart-data-series-from-rows( $series-names as xs:string*, $row* ) as chart-data-series extension item
```

where

- `$series-name` is the name of the series to be created
- `$row` is the chart-data-x-Ny-row extension item that is to be created as a series

The function returns a chart-data-series extension item, which contains the data for the X-Axis and Y-Axes of the series.

```
altova: create-chart-layer( $chart-config, $chart-data-series* ) as chart-layer extension item
```

where

- `$chart-config` is the chart-config extension item obtained with the `altova:create-chart-config` function or or via the `altova:create-chart-config-from-xml` function
- `$chart-data-series` is the chart-data-series extension item obtained with the `altova:create-chart-data-series` function or `altova:create-chart-data-series-from-rows` function

The function returns a chart-layer extension item, which contains chart-layer data.

```
altova:create-multi-layer-chart($chart-config, $chart-data-series*,
$chart-layer*)
```

where

- `$chart-config` is the chart-config extension item obtained with the `altova:create-chart-config` function or or via the `altova:create-chart-config-from-xml` function
- `$chart-data-series` is the chart-data-series extension item obtained with the `altova:create-chart-data-series` function or `altova:create-chart-data-series-from-rows` function
- `$chart-layer` is the chart-layer extension item obtained with the `altova:create-chart-layer` function

The function returns a multi-layer-chart item.

```
altova:create-multi-layer-chart($chart-config, $chart-data-series*,
$chart-layer*, xs:boolean $mergecategoryvalues)
```

where

- `$chart-config` is the chart-config extension item obtained with the `altova:create-chart-config` function or or via the `altova:create-chart-config-from-xml` function
- `$chart-data-series` is the chart-data-series extension item obtained with the `altova:create-chart-data-series` function or `altova:create-chart-data-series-from-rows` function
- `$chart-layer` is the chart-layer extension item obtained with the `altova:create-chart-layer` function

The function returns a multi-layer-chart item.

Chart Data XML Structure

Given below is the XML structure of chart data, how it might appear for the [Altova extension functions for charts](#). This affects the appearance of the specific chart. Not all elements are used for all chart kinds, e.g. the `<Pie>` element is ignored for bar charts.

Note: Chart functions are supported only in the **Enterprise and Reporting Editions** of Altova products.

```
<chart-config>
  <General
```

```

    SettingsVersion="1"    must be provided
    ChartKind="BarChart"   Pie, Pie3d, BarChart, StackedBarChart, BarChart3d,
BarChart3dGrouped, LineChart, ValueLineChart, AreaChart, StackedAreaChart, RoundGauge,
BarGauge, CandleStick
    BKColor="#ffffff"     Color
    BKColorGradientEnd="#ffffff" Color. In case of a gradient, BKColor and
BKColorGradientEnd define the gradient's colors
    BKMode="#ffffff"      Solid, HorzGradient, VertGradient
    BKFile="Path+Filename" String. If file exists, its content is drawn over the
background.
    BKFileMode="Stretch"  Stretch, ZoomToFit, Center, Tile
    ShowBorder="1"        Bool
    PlotBorderColor="#000000" Color
    PlotBKColor="#ffffff"   Color
    Title=""              String
    ShowLegend="1"        Bool
    OutsideMargin="3.%"    PercentOrPixel
    TitleToPlotMargin="3.%" PercentOrPixel
    LegendToPlotMargin="3.%" PercentOrPixel
    Orientation="vert"     Enumeration: possible values are: vert, horz
>
<TitleFont
    Color="#000000" Color
    Name="Tahoma"   String
    Bold="1"        Bool
    Italic="0"       Bool
    Underline="0"    Bool
    MinFontHeight="10. pt" FontSize (only pt values)
    Size="8.%"       FontSize />
<LegendFont
    Color="#000000"
    Name="Tahoma"
    Bold="0"
    Italic="0"
    Underline="0"
    MinFontHeight="10. pt"
    Size="3.5%" />
<AxisLabelFont
    Color="#000000"
    Name="Tahoma"
    Bold="1"
    Italic="0"
    Underline="0"
    MinFontHeight="10. pt"
    Size="5.%" />
</General>

<Line
    ConnectionShapeSize="1.%" PercentOrPixel
    DrawFilledConnectionShapes="1" Bool
    DrawOutlineConnectionShapes="0" Bool
    DrawSlashConnectionShapes="0" Bool
    DrawBackslashConnectionShapes="0" Bool
/>

<Bar
    ShowShadow="1" Bool
    ShadowColor="#a0a0a0" Color
    OutlineColor="#000000" Color

```

```

        ShowOutline="1"    Bool
    />

    <Area
        Transparency="0"    UINT ( 0-255 ) 255 is fully transparent, 0 is opaque
        OutlineColor="#000000"    Color
        ShowOutline="1"    Bool
    />

    <CandleStick
        FillHighClose="0"    Bool. If 0, the body is left empty. If 1, FillColorHighClose is used
        for the candle body
        FillColorHighClose="#ffffff"    Color. For the candle body when close > open
        FillHighOpenWithSeriesColor="1"    Bool. If true, the series color is used to fill the
        candlebody when open > close
        FillColorHighOpen="#000000"    Color. For the candle body when open > close and
        FillHighOpenWithSeriesColor is false
    />

    <Colors    User-defined color scheme: By default this element is empty except for the style
    and has no Color attributes
        UseSubsequentColors ="1"    Boolean. If 0, then color in overlay is used. If 1, then
        subsequent colors from previous chart layer is used
        Style="User"    Possible values are: "Default", "Grayscale", "Colorful", "Pastel", "User"
        Colors="#52aca0"    Color: only added for user defined color set
        Colors1="#d3c15d"    Color: only added for user defined color set
        Colors2="#8971d8"    Color: only added for user defined color set
        ...
        ColorsN=""    Up to ten colors are allowed in a set: from Colors to Colors9
    </Colors>

    <Pie
        ShowLabels="1"    Bool
        OutlineColor="#404040"    Color
        ShowOutline="1"    Bool
        StartAngle="0."    Double
        Clockwise="1"    Bool
        Draw2dHighlights="1"    Bool
        Transparency="0"    Int (0 to 255: 0 is opaque, 255 is fully transparent)
        DropShadowColor="#c0c0c0"    Color
        DropShadowSize="5.%"    PercentOrPixel
        PieHeight="10.%"    PercentOrPixel. Pixel values might be different in the result
        because of 3d tilting
        Tilt="40.0"    Double (10 to 90: The 3d tilt in degrees of a 3d pie)
        ShowDropShadow="1"    Bool
        ChartToLabelMargin="10.%"    PercentOrPixel
        AddValueToLabel="0"    Bool
        AddPercentToLabel="0"    Bool
        AddPercentToLabels_DecimalDigits="0"    UINT ( 0 - 2 )
    >
    <LabelFont
        Color="#000000"
        Name="Arial"
        Bold="0"
        Italic="0"
        Underline="0"
        MinFontHeight="10.pt"
        Size="4.%"    />
    </Pie>

```



```

<XY>
  <XAxis  Axis
    AutoRange="1"  Bool
    AutoRangeIncludesZero="1"  Bool
    RangeFrom="0."  Double: manual range
    RangeTill="1."  Double : manual range
    LabelToAxisMargin="3.%"  PercentOrPixel
    AxisLabel=""  String
    AxisColor="#000000"  Color
    AxisGridColor="#e6e6e6"  Color
    ShowGrid="1"  Bool
    UseAutoTick="1"  Bool
    ManualTickInterval="1."  Double
    AxisToChartMargin="0.px"  PercentOrPixel
    TickSize="3.px"  PercentOrPixel
    ShowTicks="1"  Bool
    ShowValues="1"  Bool
    AxisPosition="LeftOrBottom"  Enums: "LeftOrBottom", "RightOrTop",
"AtValue"
    AxisPositionAtValue = "0"  Double
  >
  <ValueFont
    Color="#000000"
    Name="Tahoma"
    Bold="0"
    Italic="0"
    Underline="0"
    MinFontHeight="10.pt"
    Size="3.%"  />
</XAxis>
  <YAxis  Axis (same as for XAxis)
    AutoRange="1"
    AutoRangeIncludesZero="1"
    RangeFrom="0."
    RangeTill="1."
    LabelToAxisMargin="3.%"
    AxisLabel=""
    AxisColor="#000000"
    AxisGridColor="#e6e6e6"
    ShowGrid="1"
    UseAutoTick="1"
    ManualTickInterval="1."
    AxisToChartMargin="0.px"
    TickSize="3.px"
    ShowTicks="1"  Bool
    ShowValues="1"  Bool
    AxisPosition="LeftOrBottom"  Enums: "LeftOrBottom", "RightOrTop",
"AtValue"
    AxisPositionAtValue = "0"  Double
  >
  <ValueFont
    Color="#000000"
    Name="Tahoma"
    Bold="0"
    Italic="0"
    Underline="0"
    MinFontHeight="10.pt"
    Size="3.%" />
</YAxis>
</XY>

```

<XY3d

AxisAutoSize="1" *Bool: If false, XSize and YSize define the aspect ration of x and y axis. If true, aspect ratio is equal to chart window*

XSize="100.%" *PercentOrPixel. Pixel values might be different in the result because of 3d tilting and zooming to fit chart*

YSize="100.%" *PercentOrPixel. Pixel values might be different in the result because of 3d tilting and zooming to fit chart*

SeriesMargin="30.%" *PercentOrPixel. Pixel values might be different in the result because of 3d tilting and zooming to fit chart*

Tilt="20." *Double. -90 to +90 degrees*

Rot="20." *Double. -359 to +359 degrees*

FoV="50."> *Double. Field of view: 1-120 degree*

>

<ZAxis

AutoRange="1"

AutoRangeIncludesZero="1"

RangeFrom="0."

RangeTill="1."

LabelToAxisMargin="3.%"

AxisLabel=""

AxisColor="#000000"

AxisGridColor="#e6e6e6"

ShowGrid="1"

UseAutoTick="1"

ManualTickInterval="1."

AxisToChartMargin="0.px"

TickSize="3.px" >

<ValueFont

Color="#000000"

Name="Tahoma"

Bold="0"

Italic="0"

Underline="0"

MinFontHeight="10.pt"

Size="3.%" />

</ZAxis>

</XY3d>**<Gauge**

MinVal="0." *Double*

MaxVal="100." *Double*

MinAngle="225" *UINT: -359-359*

SweepAngle="270" *UINT: 1-359*

BorderToTick="1.%" *PercentOrPixel*

MajorTickWidth="3.px" *PercentOrPixel*

MajorTickLength="4.%" *PercentOrPixel*

MinorTickWidth="1.px" *PercentOrPixel*

MinorTickLength="3.%" *PercentOrPixel*

BorderColor="#a0a0a0" *Color*

FillColor="#303535" *Color*

MajorTickColor="#a0c0b0" *Color*

MinorTickColor="#a0c0b0" *Color*

BorderWidth="2.%" *PercentOrPixel*

NeedleBaseWidth="1.5%" *PercentOrPixel*

NeedleBaseRadius="5.%" *PercentOrPixel*

NeedleColor="#f00000" *Color*

NeedleBaseColor="#141414" *Color*

TickToTickValueMargin="5.%" *PercentOrPixel*

MajorTickStep="10." *Double*

```

MinorTickStep="5. " Double
RoundGaugeBorderToColorRange="0.%" PercentOrPixel
RoundGaugeColorRangeWidth ="6.%" PercentOrPixel
BarGaugeRadius="5.%" PercentOrPixel
BarGaugeMaxHeight="20.%" PercentOrPixel
RoundGaugeNeedleLength="45.%" PercentOrPixel
BarGaugeNeedleLength="3.%" PercentOrPixel
>
<TicksFont
  Color="#a0c0b0"
  Name="Tahoma"
  Bold="0"
  Italic="0"
  Underline="0"
  MinFontHeight="10. pt"
  Size="4.%"
/>
<ColorRanges> User-defined color ranges. By default empty with no child element
entries
  <Entry
    From="50. " Double
    FillWithColor="1" Bool
    Color="#00ff00" Color
  />
  <Entry
    From="50.0"
    FillWithColor="1"
    Color="#ff0000"
  />
  ...
</ColorRanges>
</Gauge>
</chart-config>

```

Example: Chart Functions

The example XSLT document below shows how [Altova extension functions for charts](#) can be used. Given further below are an XML document and a screenshot of the output image generated when the XML document is processed with the XSLT document using the Altova XSLT 2.0 Engine.

Note: Chart functions are supported only in the **Enterprise and Reporting Editions** of Altova products.

Note: For more information about how chart data tables are created, see the documentation of Altova's [XMLSpy](#) and [StyleVision](#) products.

XSLT document

This XSLT document (*listing below*) uses Altova chart extension functions to generate a pie chart. It can be used to process the XML document listed further below.

```

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:altovaext="http://www.altova.com/xslt-extensions"
  exclude-result-prefixes="#all">
  <xsl:output version="4.0" method="html" indent="yes" encoding="UTF-8"/>

```

```

<xsl:template match="/">
  <html>
    <head>
      <title>
        <xsl:text>HTML Page with Embedded Chart</xsl:text>
      </title>
    </head>
    <body>
      <xsl:for-each select="/Data/Region[1]">
        <xsl:variable name="extChartConfig" as="item()*">
          <xsl:variable name="ext-chart-settings" as="item()*">
            <chart-config>
              <General>
                SettingsVersion="1"
                ChartKind="Pie3d"
                BKColor="#ffffff"
                ShowBorder="1"
                PlotBorderColor="#000000"
                PlotBKColor="#ffffff"
                Title="{@id}"
                ShowLegend="1"
                OutsideMargin="3.2%"
                TitleToPlotMargin="3.%"
                LegendToPlotMargin="6.%"
              >
              <TitleFont>
                Color="#023d7d"
                Name="Tahoma"
                Bold="1"
                Italic="0"
                Underline="0"
                MinFontHeight="10.pt"
                Size="8.%" />
              </General>
            </chart-config>
          </xsl:variable>
          <xsl:sequence select="
altovaext:create-chart-config-from-xml( $ext-chart-settings )"/>
        </xsl:variable>
        <xsl:variable name="chartDataSeries" as="item()*">
          <xsl:variable name="chartDataRows" as="item()*">
            <xsl:for-each select="( Year )"
              <xsl:sequence select="
altovaext:create-chart-data-row( (@id), ( . ) )"/>
            </xsl:for-each>
          </xsl:variable>
          <xsl:variable name="chartDataSeriesNames" as="xs:string*"
select=" ( ( &quot;Series 1&quot; ), &apos;&apos; ) [1] "/>
          <xsl:sequence
select="altovaext:create-chart-data-series-from-rows(
$chartDataSeriesNames, $chartDataRows)"/>
        </xsl:variable>
        <xsl:variable name="ChartObj" select="altovaext:create-chart(
$extChartConfig, ( $chartDataSeries ), false() )"/>
        <xsl:variable name="sChartFileName" select="' mychart1. png' "/>
        
      </xsl:for-each>
    </body>
  </html>
</xsl:template>
</xsl:stylesheet>

```

XML document

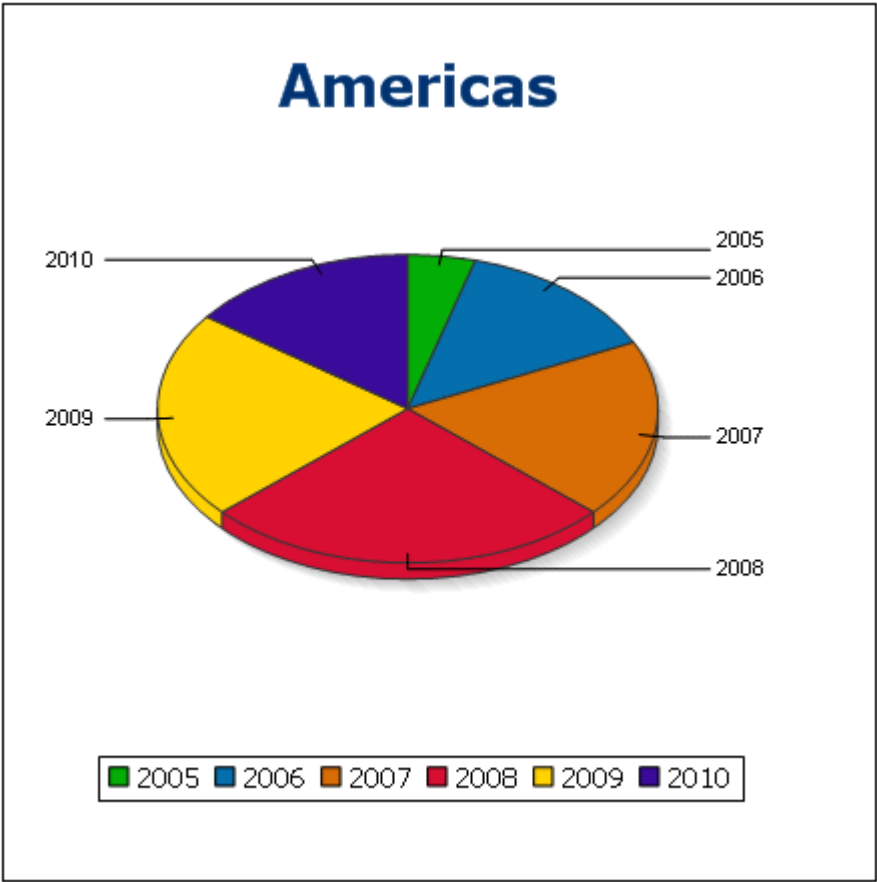
This XML document can be processed with the XSLT document above. Data in the XML

document is used to generate the pie chart shown in the screenshot below.

```
<?xml 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>
```

Output image

The pie chart show below is generated when the XML document listed above is processed with the XSLT document.



2 Datatypes in DB-Generated XML Schemas

When an XML Schema is generated from a database (DB), the datatypes specific to that DB are converted to XML Schema datatypes. The mappings of DB datatypes to XML Schema datatypes for commonly used DBs are given in this Appendix. Select from the list below.

- [MS Access](#)
- [MS SQL Server](#)
- [MySQL](#)
- [Oracle](#)
- [ODBC](#)
- [ADO](#)
- [Sybase](#)

2.1 MS Access

When an XML Schema is generated from an MS Access database (DB), the MS Access DB datatypes are converted to XML Schema datatypes as listed in the table below.

MS Access Datatype	XML Schema Datatype
GUID	xs: ID
char	xs: string
varchar	xs: string
memo	xs: string
bit	xs: boolean
Number(single)	xs: float
Number(double)	xs: double
Decimal	xs: decimal
Currency	xs: decimal
Date/Time	xs: dateTime
Number(Long Integer)	xs: integer
Number(Integer)	xs: short
Number(Byte)	xs: byte
OLE Object	xs: base64Binary

2.2 MS SQL Server

When an XML Schema is generated from an MS SQL Server database (DB), the MS SQL Server DB datatypes are converted to XML Schema datatypes as listed in the table below.

MS SQL Server Datatype	XML Schema Datatype
uniqueidentifier	xs:ID
char	xs:string
nchar	xs:string
varchar	xs:string
nvarchar	xs:string
text	xs:string
ntext	xs:string
sysname	xs:string
bit	xs:boolean
real	xs:float
float	xs:double
decimal	xs:decimal
money	xs:decimal
smallmoney	xs:decimal
datetime	xs:dateTime
smalldatetime	xs:dateTime
binary	xs:base64Binary
varbinary	xs:base64Binary
image	xs:base64Binary
integer	xs:integer
smallint	xs:short
bigint	xs:long
tinyint	xs:byte

2.3 MySQL

When an XML Schema is generated from a MySQL database (DB), the MySQL DB datatypes are converted to XML Schema datatypes as listed in the table below.

MySQL Datatype	XML Schema Datatype
char	xs:string
varchar	xs:string
text	xs:string
tinytext	xs:string
mediumtext	xs:string
longtext	xs:string
tinyint(1)	xs:boolean
float	xs:float
double	xs:double
decimal	xs:decimal
datetime	xs:dateTime
blob	xs:base64Binary
tinyblob	xs:base64Binary
mediumblob	xs:base64Binary
longblob	xs:base64Binary
smallint	xs:short
bigint	xs:long
tinyint	xs:byte

2.4 Oracle

When an XML Schema is generated from an Oracle database (DB), the Oracle DB datatypes are converted to XML Schema datatypes as listed in the table below.

Oracle Datatype	XML Schema Datatype
ROWID	xs:ID
CHAR	xs:string
NCHAR	xs:string
VARCHAR2	xs:string
NVARCHAR2	xs:string
CLOB	xs:string
NCLOB	xs:string
NUMBER (with check constraint applied) *	xs:boolean
NUMBER	xs:decimal
FLOAT	xs:double
DATE	xs:dateTime
INTERVAL YEAR TO MONTH	xs:gYearMonth
BLOB	xs:base64Binary

- * If a check constraint is applied to a column of datatype `NUMBER`, and the check constraint checks for the values 0 or 1, then the `NUMBER` datatype for this column will be converted to an XML Schema datatype of `xs:boolean`. This mechanism is useful for generating an `xs:boolean` datatype in the generated XML Schema.

2.5 ODBC

When an XML Schema is generated from an ODBC database (DB), the ODBC DB datatypes are converted to XML Schema datatypes as listed in the table below.

ODBC Datatype	XML Schema Datatype
SQL_GUID	xs:ID
SQL_CHAR	xs:string
SQL_VARCHAR	xs:string
SQL_LONGVARCHAR	xs:string
SQL_BIT	xs:boolean
SQL_REAL	xs:float
SQL_DOUBLE	xs:double
SQL_DECIMAL	xs:decimal
SQL_TIMESTAMP	xs:dateTime
SQL_DATE	xs:date
SQL_BINARY	xs:base64Binary
SQL_VARBINARY	xs:base64Binary
SQL_LONGVARBINARY	xs:base64Binary
SQL_INTEGER	xs:integer
SQL_SMALLINT	xs:short
SQL_BIGINT	xs:long
SQL_TINYINT	xs:byte

2.6 ADO

When an XML Schema is generated from an ADO database (DB), the ADO DB datatypes are converted to XML Schema datatypes as listed in the table below.

ADO Datatype	XML Schema Datatype
adGUID	xs:ID
adChar	xs:string
adWChar	xs:string
adVarChar	xs:string
adWVarChar	xs:string
adLongVarChar	xs:string
adWLongVarChar	xs:string
adVarWChar	xs:string
adBoolean	xs:boolean
adSingle	xs:float
adDouble	xs:double
adNumeric	xs:decimal
adCurrency	xs:decimal
adDBTimeStamp	xs:dateTime
adDate	xs:date
adBinary	xs:base64Binary
adVarBinary	xs:base64Binary
adLongVarBinary	xs:base64Binary
adInteger	xs:Integer
adUnsignedInt	xs:unsignedInt
adSmallInt	xs:short
adUnsignedSmallInt	xs:unsignedShort
adBigInt	xs:long
adUnsignedBigInt	xs:unsignedLong
adTinyInt	xs:byte
adUnsignedTinyInt	xs:unsignedByte

2.7 Sybase

When an XML Schema is generated from a Sybase database (DB), the Sybase DB datatypes are converted to XML Schema datatypes as listed in the table below.

Sybase Datatype	XML Schema Datatype
char	xs:string
nchar	xs:string
varchar	xs:string
nvarchar	xs:string
text	xs:string
sysname-varchar(30)	xs:string
bit	xs:boolean
real	xs:float
float	xs:float
double	xs:double
decimal	xs:decimal
money	xs:decimal
smallmoney	xs:decimal
datetime	xs:dateTime
smalldatetime	xs:dateTime
timestamp	xs:dateTime
binary<=255	xs:base64Binary
varbinary<=255	xs:base64Binary
image	xs:base64Binary
integer	xs:integer
smallint	xs:short
tinyint	xs:byte

3 Datatypes in DBs Generated from XML Schemas

When a DB structure is created from an XML Schema, the datatypes specific to that DB are generated from XML Schema datatypes. The mappings of XML Schema datatypes to DB datatypes for commonly used DBs are given in this Appendix. Select from the list below.

- [MS Access](#)
- [MS SQL Server](#)
- [MySQL](#)
- [Oracle](#)

3.1 MS Access

When an MS Access database (DB) is created from an XML Schema, the XML Schema datatypes are converted to MS Access datatypes as listed in the table below.

XML Schema Datatype	MS Access Datatype
xs:ID	GUID
xs:string	If no facets varchar (255)
	Size = either length or maxLength
	If Size <= 255 varchar (n)
	else memo
xs:normalizedString	Same as xs:string
xs:token	Same as xs:string
xs:Name	Same as xs:string
xs:NCName	Same as xs:string
xs:anyURI	Same as xs:string
xs:QName	Same as xs:string
xs:NOTATION	Same as xs:string
xs:boolean	bit
xs:float	Number (single)
xs:double	Number (double)
xs:decimal	Decimal
xs:duration	Date/Time
xs:dateTime	Date/Time
xs:time	Date/Time
xs:date	Date/Time
xs:gYearMonth	Date/Time
xs:gYear	Date/Time
xs:gMonthDay	Date/Time
xs:gDay	Date/Time
xs:gMonth	Date/Time
xs:hexBinary	If no facets varbinary (255)
	Size = either length or maxLength
	If Size <= 8000 varbinary
	else image (OLE Object)
xs:base64Binary	Same as xs:hexBinary

xs:integer	Number (Long Integer)
xs:int	Number (Long Integer)
xs:negativeInteger	Number (Long Integer); value constraint
xs:positiveInteger	Number (Long Integer); value constraint
xs:nonNegativeInteger	Number (Long Integer); value constraint
xs:nonPositiveInteger	Number (Long Integer); value constraint
xs:unsignedInt	Number (Long Integer)
xs:short	-- no equivalent --
xs:unsignedShort	-- no equivalent --
xs:long	-- no equivalent --
xs:unsignedLong	-- no equivalent --
xs:byte	Number (Byte)
xs:unsignedByte	Number (Byte)

3.2 MS SQL Server

When an XML Schema is generated from an MS SQL Server database (DB), the MS SQL Server DB datatypes are converted to XML Schema datatypes as listed in the table below.

XML Schema Datatype	MS SQL Server Datatype
ID	uniqueidentifier
xs:string	If no facets
	{ if UNICODE nvarchar (255)
	else varchar (255) }
	else
	{ if UNICODE
	(Size = either length or maxLength)
	If Size <= 4000
	if FacetLengthIsSet then nChar
	else nVarChar
	if Size <= 1073741823 then nText }
	else
	{ if NON-UNICODE
	(Size = either length or maxLength)
	If Size <= 8000
	if FacetLengthIsSet then char
	else varchar
	if Size <= 2147483647 then text }
xs:normalizedString	Same as xs:string
xs:token	Same as xs:string
xs:Name	Same as xs:string
xs:NCName	Same as xs:string
xs:anyURI	Same as xs:string
xs:QName	Same as xs:string
xs:NOTATION	Same as xs:string
xs:boolean	bit
xs:float	real
xs:double	float
xs:decimal	decimal
xs:duration	datetime

xs:dateTime	datetime
xs:time	datetime
xs:date	datetime
xs:gYearMonth	datetime
xs:gYear	datetime
xs:gMonthDay	datetime
xs:gDay	datetime
xs:gMonth	datetime
xs:hexBinary	If no facets varbinary (255)
	(Size = either length or maxLength
	If Size <= 8000
	if FacetLengthIsSet then binary
	else varbinary
	if Size <= 2147483647 then image
xs:base64Binary	Same as xs:hexBinary
xs:integer	int
xs:int	int
xs:negativeInteger	Int (constrained to {...,-2,-1})
xs:positiveInteger	Int (constrained to {1,2,...})
xs:nonNegativeInteger	int (constrained to {0,1,2,...})
xs:nonPositiveInteger	int (constrained to {...,-2,-1,0})
xs:unsignedInt	int (additional constraints)
xs:short	smallint
xs:unsignedShort	smallint (additional constraints)
xs:long	bigint
xs:unsignedLong	bigint (additional constraints)
xs:byte	tinyint
xs:unsignedByte	tinyint (additional constraints)

3.3 MySQL

When an XML Schema is generated from a MySQL database (DB), the MySQL DB datatypes are converted to XML Schema datatypes as listed in the table below.

XML Schema Datatype	MySQL Datatype
xs:ID	varchar(255)
xs:string	If no facets then varchar (255)
	else if facet length is set and <= 255 then char
	else if facet maxLength set and <= 255 then varchar
	else if maxLength is set and <= 65545 then text
	else if maxLength is set and <= 16777215 then mediumtext
	else if maxLength is set and <= 429496295 then longtext
xs:normalizedString	<i>Same as</i> xs:string
xs:token	<i>Same as</i> xs:string
xs:Name	<i>Same as</i> xs:string
xs:NCName	<i>Same as</i> xs:string
xs:anyURI	<i>Same as</i> xs:string
xs:QName	<i>Same as</i> xs:string
xs:NOTATION	<i>Same as</i> xs:string
xs:boolean	tinyint(1)
xs:float	float
xs:double	double
xs:decimal	decimal
xs:duration	timestamp
xs:dateTime	datetime
xs:time	time
xs:date	date
xs:gYearMonth	timestamp(4)
xs:gYear	year(4)
xs:gMonthDay	timestamp(8); constraints to check month, day
xs:gDay	timestamp(8); constraints to check day
xs:gMonth	timestamp(8); constraints to check month

xs:hexBinary	If no facets then blob (255)
	else if facet length is set and <= 255 then blob
	else if facet maxLength is set and <= 255 then tinyblob
	else if maxLength is set and <= 65545 then blob
	else if maxLength is set and <= 16777215 then mediumblob
	else if maxLength is set and <= 429496295 then longblob
xs:base64Binary	Same as xs:hexBinary
xs:integer	Integer
xs:int	int
xs:negativeInteger	Integer (constrained to {...,-2,-1})
xs:positiveInteger	Integer (constrained to {1,2,...})
xs:nonNegativeInteger	Integer (constrained to {0,1,2,...})
xs:nonPositiveInteger	Integer (constrained to {...,-2,-1,0})
xs:unsignedInt	Int (additional constraints)
xs:short	Smallint
xs:unsignedShort	Smallint (additional constraints)
xs:long	Bigint
xs:unsignedLong	Bigint (additional constraints)
xs:byte	Tinyint
xs:unsignedByte	Tinyint (additional constraints)

3.4 Oracle

When an XML Schema is generated from an Oracle database (DB), the Oracle DB datatypes are converted to XML Schema datatypes as listed in the table below.

XML Schema Datatype	Oracle Datatype
xs:ID	ROWID
xs:string	If no facets
	if UNICODE then NVARCHAR2 (255)
	else VARCHAR2 (255)
	else if UNICODE
	(Size = either length or maxLength)
	If Size <= 2000 then NCHAR
	if Size <= 4000 then NVARCHAR2
	if Size <= 4 Gigabytes then NCLOB
	else if NON-UNICODE
	(Size = either length or maxLength)
	If Size <= 2000 then CHAR
	if Size <= 4000 then VARCHAR2
	if Size <= 4 Gigabytes then CLOB
xs:normalizedString	Same as xs:string
xs:token	Same as xs:string
xs:Name	Same as xs:string
xs:NCName	Same as xs:string
xs:anyURI	Same as xs:string
xs:QName	Same as xs:string
xs:NOTATION	Same as xs:string
xs:boolean	NUMBER with constraint Boolean
xs:float	FLOAT
xs:double	FLOAT
xs:decimal	NUMBER
xs:duration	TIMESTAMP
xs:dateTime	TIMESTAMP
xs:time	DATE
xs:date	DATE
xs:gYearMonth	INTERVAL YEAR TO MONTH

xs:gYear	DATE
xs:gMonthDay	DATE
xs:gDay	DATE
xs:gMonth	DATE
xs:hexBinary	if no facets then RAW (255)
	(Size = either length or maxLength)
	If Size <= 2000 then RAW (X)
	else Size <= 2 Gigabytes then LONG RAW (X)
	if Size <= 4 Gigabytes then BLOB (X)
xs:base64Binary	BLOB
xs:integer	NUMBER
xs:int	NUMBER
xs:negativeInteger	NUMBER (constrained to {...,-2,-1})
xs:positiveInteger	NUMBER (constrained to {1,2,...})
xs:nonNegativeInteger	NUMBER (constrained to {0,1,2,...})
xs:nonPositiveInteger	NUMBER (constrained to {...,-2,-1,0})
xs:unsignedInt	NUMBER (additional constraints)
xs:short	NUMBER
xs:unsignedShort	NUMBER (additional constraints)
xs:long	NUMBER
xs:unsignedLong	NUMBER (additional constraints)
xs:byte	BLOB
xs:unsignedByte	BLOB (additional constraints)

4 Technical Data

This section contains useful background information on the technical aspects of your software. It is organized into the following sections:

- [OS and Memory Requirements](#)
- [Altova XML Parser](#)
- [Altova XSLT and XQuery Engines](#)
- [Unicode Support](#)
- [Internet Usage](#)

4.1 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.

4.2 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.

4.3 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.

4.4 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.

4.4.1 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.

4.4.2 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.

4.5 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.

5 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.

5.1 Electronic Software Distribution

This product is available through electronic software distribution, a distribution method that provides the following unique benefits:

- You can evaluate the software free-of-charge 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.

5.2 Software Activation and License Metering

As part of Altova's Software Activation, the software may use your internal network and Internet connection for the purpose of transmitting license-related data at the time of installation, registration, use, or update to an Altova-operated license server and validating the authenticity of the license-related data in order to protect Altova against unlicensed or illegal use of the software and to improve customer service. Activation is based on the exchange of license related data such as operating system, IP address, date/time, software version, and computer name, along with other information between your computer and an Altova license server.

Your Altova product has a built-in license metering module that further helps you avoid any unintentional violation of the End User License Agreement. Your product is licensed either as a single-user or multi-user installation, and the license-metering module makes sure that no more than the licensed number of users use the application concurrently.

This license-metering technology uses your local area network (LAN) to communicate between instances of the application running on different computers.

Single license

When the application starts up, as part of the license metering process, the software sends a short broadcast datagram to find any other instance of the product running on another computer in the same network segment. If it doesn't get any response, it will open a port for listening to other instances of the application.

Multi 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 \(http://www.iana.org/\)](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.

5.3 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.

5.4 Altova End User License Agreement

THIS IS A LEGAL DOCUMENT -- RETAIN FOR YOUR RECORDS

ALTOVA® END USER LICENSE AGREEMENT

Licensors:
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 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 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, 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, Rudolfplatz, 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

Index

■

.docx, 181, 467**.NET,**

- and XMLSpy Debuggers, 544
- differences to XMLSpy standalone, 542
- integration of XMLSpy with, 539

.NET extension functions,

- constructors, 1402
- datatype conversions, .NET to XPath/XQuery, 1405
- datatype conversions, XPath/XQuery to .NET, 1404
- for XSLT and XQuery, 1400
- instance methods, instance fields, 1403
- overview, 1400
- static methods, static fields, 1402

.pptx, 181, 467**.xlsx, 181, 467**

2

2005R3,

- compatibility mode, 596

A

Activating the software, 921**Active configuration,**

- for global resources, 885

ActiveX controls,

- support, 970

Add Child command,

- in Grid View, 695

Adding,

- overlay charts, 237

ADO,

- conversion of datatypes in XML Schema generation from DB, 1429

ADO Connections, 482, 775**Alias,**

- see Global Resources, 495

Altova Engines,

- in Altova products, 1443

Altova extension functions,

- chart functions (see chart functions), 1408
- general functions, 1408

Altova extensions,

- chart functions (see chart functions), 1407

Altova Global Resources,

- see under Global Resources, 495

Altova products, 31**Altova Scripting Projects, 931****Altova support, 31****Altova XML Parser,**

- about, 1442

Altova XSLT 1.0 Engine,

- limitations and implementation-specific behavior, 1380

Altova XSLT 2.0 Engine,

- general information about, 1382
- information about, 1382

API,

- accessing, 1357
- documentation, 985
- JAVA, 1297
- JAVA Classpath, 1297
- overview, 987

Append,

- row (in Authentic View), 768

Append command,

- in Grid View, 688

Appending,

- overlay charts, 237

Application,

- ActiveDocument, 1031
- AddMacroMenuItem, 1031
- Application, 1031
- ClearMacroMenu, 1031
- CurrentProject, 1032
- Dialogs, 1032
- Documents, 1033
- GetDatabaseImportElementList, 1033
- GetDatabaseSettings, 1034
- GetDatabaseTables, 1034
- GetExportSettings, 1035
- GetTextImportElementList, 1035
- GetTextImportExportSettings, 1036
- ImportFromDatabase, 1036
- ImportFromSchema, 1037

Application,

- ImportFromText, 1038
- ImportFromWord, 1039
- NewProject, 1040
- OnBeforeOpenDocument, 1029
- OnBeforeOpenProject, 1029
- OnDocumentOpened, 1030
- OnProjectOpened, 1030
- OpenProject, 1040
- Parent, 1040
- Quit, 1041
- ReloadSettings, 1041
- RunMacro, 1041
- ScriptingEnvironment, 1042
- ShowApplication, 1042
- ShowForm, 1043
- URLDelete, 1043
- URLMakeDirectory, 1044
- WarningNumber, 1044
- WarningText, 1044

Application Events, 948**Application name, 596****Application-level,**

- integration of XMLSpy, 1331

Apply, 900**Archive View, 181, 467**

- and EPUB files, 474
- and OOXML files, 469
- and ZIP files, 472

Arcroles in XBRL, 858**area chart features, 226****ASPRJ files, 931****Assign,**

- shortcut to a command, 888

Assigning StyleVision Power Stylesheet to XML file, 763**ATL,**

- plug-in sample files, 974

atomization of nodes,

- in XPath 2.0 and XQuery 1.0 evaluation, 1388

ATTLIST declaration,

- add as child in Grid View, 700
- appending in Grid View, 694
- convert to in Grid View, 702
- inserting in Grid View, 687

Attribute,

- add as child in Grid View, 696
- appending in Grid View, 689

- convert to in Grid View, 701
- in schema definitions, 59
- inserting in Grid View, 682
- toggle in Content model view, 59

Attribute preview, 904**Attribute values,**

- entering in Authentic View, 374

AttributeFormDefault,

- settings in Schema Design View, 729

Attributes entry helper,

- in Authentic View, 174

Attributes of schema components,

- defining in Schema View, 115

Authentic menu, 761

- dynamic table editing, 169
- markup display, 169

Authentic Scripting,

- security settings, 769
- trusted locations, 769

Authentic View,

- adding nodes, 369
- applying elements, 369
- CDATA sections in, 372
- clearing elements, 369
- context menu, 367
- context menus, 178
- data entry devices in, 372
- displaying markup tags, 367
- document display, 171
- editing data in an XML DB, 764
- editing DB data in, 762
- editing XML in, 193
- entering attribute values, 374
- entering data in, 372
- entities in, 372
- entry helpers, 367
- entry helpers in, 174
- formatting text in, 169
- inserting entities in, 374
- inserting nodes, 369
- interface, 168
- main window in, 171
- markup display in, 169, 171
- opening an XML document in, 366
- opening new XML file in, 761
- overview of GUI, 168
- paste as XML/Text, 178
- printing an XML document from, 375

Authentic View,

- removing nodes, 369
- special characters in, 372
- SPS Tables, 382
- switching to, 834
- tables (SPS and XML), 381
- tables in, 369
- toolbar icons, 169
- tutorial, 365
- usage of important features, 377
- usage of XML tables, 383
- XML table icons, 387
- XML tables, 383

Authentic View Events, 948**Authentic View template, 366****Authentic XML, 363****AuthenticDataTransfer,**

- dropEffect, 1047
- getData, 1047
- ownDrag, 1047
- type, 1047

AuthenticRange,

- AppendRow, 1052
- Application, 1052
- CanPerformAction, 1052
- CanPerformActionWith, 1053
- Close, 1053
- CollapsToBegin, 1054
- CollapsToEnd, 1054
- Copy, 1054
- Cut, 1054
- Delete, 1055
- DeleteRow, 1055
- DuplicateRow, 1055
- ExpandTo, 1056
- FirstTextPosition, 1057
- FirstXMLData, 1057
- FirstXMLDataOffset, 1058
- GetElementAttributeNames, 1059
- GetElementAttributeValue, 1059
- GetElementHierarchy, 1060
- GetEntityNames, 1061
- Goto, 1061
- GotoNext, 1062
- GotoNextCursorPosition, 1062
- GotoPrevious, 1063
- GotoPreviousCursorPosition, 1063
- HasElementAttribute, 1064

- InsertEntity, 1064

- InsertRow, 1065

- IsCopyEnabled, 1065

- IsCutEnabled, 1065

- IsDeleteEnabled, 1066

- IsEmpty, 1066

- IsEqual, 1066

- IsFirstRow, 1066

- IsInDynamicTable, 1067

- IsLastRow, 1067

- IsPasteEnabled, 1067

- IsTextStateApplied, 1068

- LastTextPosition, 1068

- LastXMLData, 1069

- LastXMLDataOffset, 1069

- MoveBegin, 1070

- MoveEnd, 1071

- MoveRowDown, 1071

- MoveRowUp, 1071

- Parent, 1072

- Paste, 1072

- PerformAction, 1072

- Select, 1073

- SelectNext, 1073

- SelectPrevious, 1074

- SetElementAttributeValue, 1075

- SetFromRange, 1076

- Text, 1076

AuthenticView, 1093

- Application, 1086

- AsXMLString, 1087

- DocumentBegin, 1088

- DocumentEnd, 1088

- Event, 1089

- Goto, 1090

- IsRedoEnabled, 1091

- IsUndoEnabled, 1091

- MarkupVisibility, 1091

- OnBeforeCopy, 1078

- OnBeforeCut, 1078

- OnBeforeDelete, 1079

- OnBeforeDrop, 1079

- OnBeforePaste, 1080

- OnDragOver, 1081

- OnKeyboardEvent, 1082

- OnMouseEvent, 1083

- OnSelectionChanged, 1084

- Parent, 1091

AuthenticView, 1093

- Print, 1092
- Redo, 1092
- Selection, 1092
- Undo, 1093
- WholeDocument, 1094
- XMLDataRoot, 1094

Auto-complete,

- text view enable/disable, 903

Auto-completion in SQL scripts, 793**Auto-hiding windows, 11****Auto-Macro setting, 951****Automatic validation, 902**

B

Back,

- in Schema View, 145

Background,

- gradient, 596
- status updates, 533
- status updates - increase interval, 519

Background Information, 1440**backwards compatibility,**

- of XSLT 2.0 Engine, 1382

bar chart features, 226**Base package, 596****Base type,**

- modifying, 131

Big-endian, 912**Bookmark, 920****Bookmark margin, 836****Bookmarks,**

- inserting and removing, 651
- navigating, 651

Bookmarks in SQL scripts, 793**Bookmarks in Text View, 100****Breakpoint,**

- dialog box, 759

Breakpoints,

- using in SOAP debugger, 432
- using in XSLT/XQuery Debugger, 341

Breakpoints dialog, 329**Browse,**

- Oracle XML Db, 809

Browser, 904

- View, 834

Browser menu, 838**Browser pane,**

- in Database Query window, 789

Browser View, 180, 838

- back, 838
- font size, 838
- forward, 838
- refresh content, 838
- separate window, 838
- stop loading page, 838

C

C#,

- code, 555
- integration of XMLSpy, 1335, 1336
- settings, 596

C++,

- code, 555
- settings, 596

Call Stack Window,

- in XSLT/XQuery Debugger, 338

Callgraph profiling, 350**Calling named templates, 311****candlestick chart features, 226****Carriage return key,**

- see Enter key, 400

Cascade,

- Window, 917

Catalog,

- Oasis XML, 707

Catalogs, 278**CDATA,**

- add as child in Grid View, 696
- appending in Grid View, 690
- convert to in Grid View, 701
- inserting in Grid View, 683

CDATA sections,

- inserting in Authentic View, 377

Certificate stores, 257**Certificates, 257****Changing view,**

- to Authentic View, 169

Chapters, 920**Character,**

Character,

position, 836

character entities,

in HTML output of XSLT transformation, 1380

character normalization,

in XQuery document, 1385

Character-Set,

encoding, 912

Chart data table,

how it is constructed, 201

Chart functions,

chart data structure for, 1414

example, 1419

listing, 1411

Charts,

3d settings, 234

adding legend, 225

appearance, 224

area chart features, 226

background color, 225

bar chart features, 226

candlestick chart features, 226

chart data, 216

chart settings, 217

color range, 229

color schema, 229

defining colors, 229

example (advanced), 240

example (candlestick), 246

example (simple), 239

exporting, 238

fonts, 236

gauge chart features, 226

grid lines, 231, 233, 234

in XSLT/XQuery Profiler, 360

line chart features, 226

margins, 235

multiple tabs for, 199

overlays, 237

overview, 199

pie chart features, 226

reloading, 199

removing legend, 225

series color, 229

sizes, 235

Source XPath, 205

tick size, 235

title, 225

Type, 219

X-axis, 231

X-Axis selection, 208

Y-axis, 233

Y-Axis selection, 212

Z-axis, 234

Charts Window, 25**Check,**

spelling checker, 869

Class,

JAVA, 1297

Class ID,

in XMLSpy integration, 1331

ClassPath statement, 1297**Code,**

built in types, 608

SPL, 597

Code compatibility,

v2005R3, 596

Code Generator, 555, 727**Code page, 906****CodeGeneratorDlg,**

Application, 1095

CPPSettings_DOMType, 1096

CPPSettings_LibraryType, 1097

CPPSettings_UseMFC, 1097

CSharpSettings_ProjectType, 1098

OutputPath, 1098

OutputPathDialogAction, 1098

OutputResultDialogAction, 1098

Parent, 1099

ProgrammingLanguage, 1099

PropertySheetDialogAction, 1099

TemplateFileName, 1100

Collapse,

unselected, 835

collations,

in XPath 2.0, 1388

in XQuery document, 1385

Color, 906, 909

tab, 910

table, 910

COM-API,

documentation, 985

Command,

add to toolbar/menu, 886

context menu, 893

delete from menu, 893

- Command,**
 - reset menu, 893
- Command line, 924**
- Commands,**
 - listing in key map, 921
- Comment,**
 - add as child in Grid View, 696
 - appending in Grid View, 690
 - convert to in Grid View, 702
 - inserting in Grid View, 683
- Commenting in and out,**
 - in XML documents in Text View., 188
- Commenting XML text in and out, 651**
- Comments in SQL scripts, 793**
- Communication process,**
 - SOAP debugger, 425
- Comparing directories, 880**
- Comparing files, 878**
 - options, 883
- Comparisons,**
 - of directories, 518
 - of files, 517
 - of files and directories, 516
- Compatibility mode,**
 - v2005R3, 596
- Compile,**
 - for Linux/GCC, 560
- Compiler,**
 - settings, 596
- Complex type,**
 - extending definition, 50
 - in schema definitions, 50
- Component definition,**
 - reusing, 50
- Component Navigator, 132**
- Compositor,**
 - for sequences, 39
 - in Schema View, 120
- Configurations,**
 - of a global resource, 496
- Configurations in global resources, 508**
- Configure,**
 - XMLSPY UI, 971
- Configure view,**
 - dialog for Content Model View, 736
- Connecting to data source, 479, 480, 482, 486, 771, 772, 775, 779**
 - via a global resource, 492, 785
- Connecting to SchemaAgent Server, 283, 743**
- Connection Wizard, 480, 772**
- Constraints, 135**
- Content Model,**
 - creating a basic model, 39
 - save diagram, 731
 - toggle attributes, 59
- Content Model View, 35, 120**
 - configuring, 736
- Content models,**
 - of schema components, 120
- Context menu,**
 - commands, 893
 - for customization, 898
- Context menus,**
 - in Authentic View, 178
- Context Window,**
 - in XSLT/XQuery Debugger, 337
- Convert,**
 - database data to XML, 813
 - database schema to XML Schema, 818
 - MS Word data to XML, 817
 - Oracle XML Db, 806
 - schema to DB structure, 823
 - text file to XML, 811
- Convert menu, 811**
- Convert To command,**
 - in Grid View, 701
- Copy,**
 - as XML text, 637
 - XML as structured text, 638
- Copy command, 637**
- Copy XPath, 640**
- Copy XPointer, 640**
- Copyright information, 923, 1447**
- count() function,**
 - in XPath 1.0, 1380
- count() function in XPath 2.0,**
 - see fn:count(), 1388
- CPU,**
 - load - increase background status updates, 519
- CR&LF, 900**
- Create,**
 - DB based on schema, 823
- CSS, 403**
 - auto-completion, 406
 - document outline, 406
 - Info window, 406

CSS, 403

- properties, 406
- syntax coloring, 406

CSS Info window, 406**CSV file,**

- import as XML, 811

Custom dictionary, 869**CustomCatalog, 707****Customization, 29****Customize,**

- context menu, 893
- Customize context menu, 898
- macros, 895
- menu, 893
- toolbar/menu commands, 886

Cut command, 637**CVS, 519**

D

Database,

- ADO connections, 482, 775
- connecting to, 479, 771
- connecting via global resource, 492, 785
- Connection Wizard, 480, 772
- create DB based on schema, 823
- editing records of, 797
- Existing connections, 481, 774
- export of XML data to, 829
- import data as XML, 813
- import structure as XML Schema, 818
- ODBC connections, 486, 779
- Oracle XML Db, 806

Database Query,

- Browser pane in DB Query window, 789
- Connecting to DB for query, 788
- creating the query, 796
- Editing results, 797
- Messages pane, 797
- Results of, 797

Database Query window, 786**Database/Table View,**

- how to use, 83

DatabaseConnection,

- ADOConnection, 1101
- AsAttributes, 1101

CreateMissingTables, 1102

CreateNew, 1102

DatabaseKind, 1102

ExcludeKeys, 1103

File, 1103

IncludeEmptyElements, 1104

NumberDateTimeFormat, 1104

ODBCConnection, 1104

SQLSelect, 1105

TextFieldLen, 1106

Databases,

- and global resources, 507
- editing in Authentic View, 762
- see also DB, 389
- support in XMLSpy, 494

Databases in XMLSpy, 478**DatabaseSpy,**

- 3d charts, 234
- area chart features, 226
- bar chart features, 226
- candlestick chart features, 226
- chart background, 225
- chart colors, 229
- chart font options, 236
- chart fonts, 236
- chart grid, 231, 233, 234
- chart legend, 225
- chart title, 225
- chart X-axis, 231
- chart Y-axis, 233
- chart Z-axis, 234
- charts sizes, 235
- gauge chart features, 226
- line chart features, 226
- pie chart features, 226

Datatypes,

- conversion of DB to XML Schema, 1423
- for DBs generated from XML Schemas, 1431
- in XPath 2.0 and XQuery 1.0, 1388
- see also XML Schema datatypes, 1424

Date Picker,

- using in Authentic View, 395

Dates,

- changing manually, 396

DB,

- creating queries, 390
- datatype conversion for XML Schema, 1423
- datatypes from XML Schema, 1431

DB,

- editing in Authentic View, 389, 394
- filtering display in Authentic View, 390
- navigating tables in Authentic View, 389
- parameters in DB queries, 390
- queries in Authentic View, 389

DB XML,

- assigning XML Schemas to, for IBM DB2, 802
- managing XML Schemas, for IBM DB2, 799

db2-fn:sqlquery, 323**db2-fn:xmlicolumn, 323****Debugger,**

- breakpoints/tracepoints dialog box, 759
- debug windows, 760
- enable/disable breakpoint, 758
- enable/disable tracepoint, 759
- end session, 757
- for SOAP, 855
- insert/remove breakpoint, 758
- insert/remove tracepoint, 758
- options for SOAP, 857
- restart XSLT Debugger, 757
- settings, 760
- show curr. exec. nodes, 758
- SOAP communication example, 426
- start XSLT Debugger, 756
- step into, 757
- step out, 757
- step over, 757
- stop XSLT Debugger, 757

Debugger windows,

- arrangement, 340

Debugging macros, 955**deep-equal() function in XPath 2.0,**

- see fn:deep-equal(), 1388

Default,

- encoding, 912
- menu, 893

Default editor, 902**default functions namespace,**

- for XPath 2.0 and XQuery 1.0 expressions, 1388
- in XSLT 2.0 stylesheets, 1382

Default view,

- setting in Main Window, 902

Defining,

- 3d settings, 234
- area chart features, 226
- bar chart features, 226

candlestick chart features, 226

chart fonts, 236

charts colors, 229

charts sizes, 235

charts title, 225

charts type, 219

color of charts, 229

fonts in charts, 236

gauge chart features, 226

grid lines, 231, 233, 234

line chart features, 226

pie chart features, 226

X-axis settings, 231

Y-axis settings, 233

Z-axis settings, 234

Delete,

- Application.URLDelete, 1043
- command from context menu, 893
- command from toolbar, 886
- icon from toolbar, 886
- row (in Authentic View), 768
- shortcut, 888
- toolbar, 887

Delete command, 637**Deleting,**

- overlay charts, 237

Derived types,

- modifying base type of, 131

Deriving a schema type, 139**Details Entry Helper, 39****Dialogs,**

- Application, 1107
- CodeGeneratorDlg, 1107
- DTDSchemaGeneratorDlg, 1108
- FileSelectionDlg, 1107
- GenerateSampleXMLDlg, 1108
- Parent, 1107
- SchemaDocumentationDlg, 1108

Dictionary,

- adding custom, 869
- modifying existing, 869
- spelling checker, 869

Diffdog,

- configure for differencing, 533

Differencing,

- configuring Diffdog, 533

directories,

- comparing two, 880

directories,

creating with Application.URLMakeDirectory, 1044

Directory comparisons, 516, 518**Disable,**

breakpoint - XSLT debugger, 758

tracepoint - XSLT debugger, 759

Disconnect XMLSpy from SchemaAgent, 744**Display all globals, 740****Display diagram, 740****Display Settings, 910****Distribution,**

of Altova's software products, 1447, 1448, 1450

DII,

compiler settings, 596

DocEditEvent (obsolete),

altKey (obsolete), 1239

altLeft (obsolete), 1240

button (obsolete), 1241

cancelBubble (obsolete), 1242

clientX (obsolete), 1243

clientY (obsolete), 1244

ctrlKey (obsolete), 1245

ctrlLeft (obsolete), 1246

dataTransfer (obsolete), 1247

fromElement (obsolete), 1248

keyCode (obsolete), 1248

propertyName (obsolete), 1249

repeat (obsolete), 1249

returnValue (obsolete), 1249

shiftKey (obsolete), 1250

shiftLeft (obsolete), 1251

srcElement (obsolete), 1252

type (obsolete), 1253

DocEditView (obsolete),

ApplyTextState (obsolete), 1259

CurrentSelection (obsolete), 1260

EditClear (obsolete), 1260

EditCopy (obsolete), 1261

EditCut (obsolete), 1261

EditPaste (obsolete), 1262

EditRedo (obsolete), 1262

EditSelectAll (obsolete), 1263

EditUndo (obsolete), 1263

event (obsolete), 1264

GetAllowedElements (obsolete), 1264

GetNextVisible (obsolete), 1267

GetPreviousVisible (obsolete), 1268

IsEditClearEnabled (obsolete), 1269

IsEditCopyEnabled (obsolete), 1269

IsEditCutEnabled (obsolete), 1270

IsEditPasteEnabled (obsolete), 1270

IsEditRedoEnabled (obsolete), 1271

IsEditUndoEnabled (obsolete), 1271

IsRowAppendEnabled (obsolete), 1272

IsRowDeleteEnabled (obsolete), 1272

IsRowDuplicateEnabled (obsolete), 1273

IsRowInsertEnabled (obsolete), 1273

IsRowMoveDownEnabled (obsolete), 1274

IsRowMoveUpEnabled (obsolete), 1274

IsTextStateApplied (obsolete), 1275

IsTextStateEnabled (obsolete), 1275

LoadXML (obsolete), 1276

RowAppend (obsolete), 1278

RowDelete (obsolete), 1278

RowDuplicate (obsolete), 1279

RowInsert (obsolete), 1279

RowMoveDown (obsolete), 1280

RowMoveUp (obsolete), 1280

SaveXML (obsolete), 1281

SelectionMoveTabOrder (obsolete), 1282

SelectionSet (obsolete), 1283

XMLRoot (obsolete), 1284

Dockable window, 917**Docking windows, 11****DOCTYPE declaration,**

add as child in Grid View, 699

appending in Grid View, 693

convert to in Grid View, 702

inserting in Grid View, 685

Document,

Application, 1114

AssignDTD, 1114

AssignSchema, 1114

AssignXSL, 1115

AssignXSLFO, 1115

AuthenticView, 1116

browse Oracle XML Db, 809

Close, 1116

ConvertDTDOOrSchema, 1117

CreateChild, 1118

CreateSchemaDiagram, 1119

CurrentViewMode, 1119

DataRoot, 1119

DocEditView, 1120

Encoding, 1120

EndChanges, 1120

Document,

- ExecuteXQuery, 1121
- ExportToDatabase, 1121
- ExportToText, 1122
- FullName, 1123
- GenerateDTDOOrSchema, 1123, 1124
- GenerateProgramCode, 1124
- GenerateSampleXML, 1124
- GenerateSchemaDocumentation, 1124
- GetExportElementList, 1126
- GetPathName, 1127
- GridView, 1127
- IsModified, 1128
- IsValid, 1128
- IsWellFormed, 1129
- Name, 1130
- OnBeforeCloseDocument, 1112
- OnBeforeSaveDocument, 1111
- OnBeforeValidate, 1112
- OnCloseDocument, 1113
- OnViewActivation, 1113
- Path, 1130
- RootElement, 1130
- Save, 1131
- SaveAs, 1131
- Saved, 1131
- SaveInString, 1131
- SaveToURL, 1132
- SetActiveDocument, 1132
- SetEncoding, 1133
- SetExternalsIsValid, 1134
- SetPathName, 1134
- Spelling checker, 869
- StartChanges, 1134
- SwitchViewMode, 1135
- Title, 1135
- TransformXSL, 1136
- TransformXSLFO, 1136
- UpdateViews, 1137
- UpdateXMLData, 1137
- XQuery, 1121

Document Events, 948**Documentation,**

- for schema, 65
- of WSDL files, 845
- of XBRL taxonomies, 864
- of XML Schema files, 731

Document-level,

- examples of integration of XMLSpy, 1335
- integration of XMLSpy, 1334, 1335
- integration of XMLSpyControl, 1334

Documents,

- Count, 1138
- Item, 1138
- NewAuthenticFile, 1139
- NewFile, 1139
- NewFileFromText, 1139
- OpenAuthenticFile, 1140
- OpenFile, 1140
- OpenURL, 1140
- OpenURLDialog, 1141

Documents in Main Window, 12**DTD,**

- assigning to XML document, 718
- Attlist declaration in, 687, 694, 700, 702
- converting to UML, 724
- converting to XML Schema, 722
- Element declaration in, 687, 694, 699, 702
- Entity declaration in, 687, 694, 700, 702
- generate outline XML file from, 726
- generating code from, 727
- generating from XML document, 720
- generating from XML Schema (Enterprise and Professional editions), 266
- go to definition in from XML document, 720
- go to from XML document, 719
- including entities, 719
- menu commands related to, 718
- Notation declaration in, 687, 694, 700, 703
- reference from XML to external DTD, 686

DTD/Schema menu, 718**DTDs, 263, 900, 902**

- converting to XML Schemas (Enterprise and Professional editions), 264
- editing in Grid View (Enterprise and Professional editions), 264
- editing in Text View, 264
- generating XML document from, 264

DTDSchemaGeneratorDlg,

- Application, 1142
- AttributeTypeDefinition, 1142
- DTDSchemaFormat, 1142
- FrequentElements, 1143
- GlobalAttributes, 1143
- MaxEnumLength, 1143
- MergeAllEqualNamed, 1143

DTDSchemaGeneratorDlg,

- OnlyStringEnums, 1144
- OutputPath, 1144
- OutputPathDialogAction, 1144
- Parent, 1144
- ResolveEntities, 1144
- TypeDetection, 1145
- ValueList, 1145

Duplicate,

- row (in Authentic View), 768

Dynamic (SPS) tables in Authentic View,

- usage of, 382

Dynamic tables,

- editing, 169

E

Eclipse platform,

- and XMLSpy, 546
- and XMLSpy Integration Package, 547
- XMLSpy Debugger perspectives, 554
- XMLSpy perspective in, 551

Edit,

- macro button, 898

Edit menu, 636**Edited with XMLSPY, 900****Editing database records, 797****Editing in Text View, 103****Editing views, 97****Element,**

- add as child in Grid View, 696
- appending in Grid View, 689
- convert to in Grid View, 701
- inserting in Grid View, 682
- making optional, 46
- restricting content, 46

ELEMENT declaration,

- add as child in Grid View, 699
- appending in Grid View, 694
- convert to in Grid View, 702

ELEMENT declaration in DTD,

- inserting in Grid View, 687

element type,

- specifying in XML document, 71

ElementFormDefault,

- settings in Schema Design View, 729

ElementList,

- Count, 1146
- Item, 1146
- RemoveElement, 1146

ElementListItem,

- ElementKind, 1146
- FieldCount, 1147
- Name, 1147
- RecordCount, 1147

Elements entry helper,

- in Authentic View, 174

E-mail,

- sending files with, 632

Empty elements, 902**Empty lines,**

- in XML documents in Text View, 188

Enable breakpoint - XSLT debugger, 758**Enable tracepoint - XSLT debugger, 759****Enclose in Element command, 706****encoding,**

- default, 912
- in XQuery document, 1385
- of files, 626

End,

- debugger session, 757

End User License Agreement, 1447, 1451**End-of-line markers, 836****Engine information, 1379****Enhanced Grid View, 833**

- see Grid View, 73

Enter key,

- effects of using, 400

Entities,

- defining in Authentic View, 377, 396
- in XML Schema-based XML, 186
- inserting in Authentic View, 374, 377

Entities entry helper,

- in Authentic View, 174

ENTITY declaration,

- add as child in Grid View, 700
- appending in Grid View, 694
- convert to in Grid View, 702
- inserting in Grid View, 687

Entry Helper, 34

- Details, 39
- in Grid View, 81

Entry helpers, 15

- for XML documents, 195

Entry helpers, 15

- for XQuery, 319
- in Schema View, 132
- toggling display on and off, 918
- updating, 711

Entry helpers in Text View, 105**Entry-Helper, 918****Enumeration,**

- defining for attributes, 59

Enumerations,

- in XMLSpyControl, 1374
- SPYAttributeTypeDefinition, 1286
- SPYAuthenticActions, 1286
- SPYAuthenticDocumentPosition, 1286
- SPYAuthenticElementActions, 1287
- SPYAuthenticElementKind, 1287
- SPYAuthenticMarkupVisibility, 1287
- SPYDatabaseKind, 1288
- SPYDialogAction, 1288
- SPYDOMType, 1288
- SPYDTDSchemaFormat, 1288
- SPYEncodingByteOrder, 1289
- SPYExportNamespace, 1289
- SPYFrequentElements, 1289
- SPYKeyEvent, 1290
- SPYLibType, 1290
- SPYLoading, 1290
- SPYMouseEvent, 1290
- SPYNumberDateTimeFormat, 1291
- SPYProgrammingLanguage, 1291
- SPYProjectItemTypes, 1292
- SPYProjectType, 1292
- SPYSampleXMLGenerationOptimization, 1292
- SPYSampleXMLGenerationSchemaOrDTDAssignment, 1292
- SPYSchemaDefKind, 1292
- SPYSchemaDocumentationFormat, 1293
- SPYTextDelimiters, 1294
- SPYTextEnclosing, 1294
- SPYTypeDetection, 1294
- SPYURLTapes, 1294
- SPYViewModes, 1294
- SPYVirtualKeyMask, 1295
- SPYXMLDataKind, 1295

EPUB files, 474**Evaluating XPath, 17****Evaluation key,**

- for your Altova software, 921

Evaluation period,

- of Altova's software products, 1447, 1448, 1450

Event, 1029, 1030, 1078, 1079, 1080, 1081, 1082, 1083, 1084, 1111, 1112, 1113, 1165, 1166, 1167**Event handlers,**

- in Scripting Project, 948
- overview, 936

Events, 948, 994

- and event handlers, 938

Example files,

- tutorial, 33

Examples,

- location of installed files, 31

Excel 2007, 181, 467**Exception,**

- out of memory, 594

Expand,

- fully, 835

Explorer, 902**Exporting XML data to database, 829****Exporting XML data to text files, 826****ExportSettings,**

- CreateKeys, 1148
- ElementList, 1148
- EntitiesToText, 1148
- ExportAllElements, 1148
- FromAttributes, 1149
- FromSingleSubElements, 1149
- FromTextValues, 1149
- IndependentPrimaryKey, 1149
- Namespace, 1149
- SubLevelLimit, 1150

Extended schema validation, 271**Extended validation, 290**

- in SchemaAgent, 745

Extension functions for XSLT and XQuery, 1392**Extension Functions in .NET for XSLT and XQuery,**

- see under .NET extension functions, 1400

Extension Functions in Java for XSLT and XQuery,

- see under Java extension functions, 1392

Extension Functions in MSXSL scripts, 1405**external functions,**

- in XQuery document, 1385

External ID declaration,

- add as child in Grid View, 699
- appending in Grid View, 693
- convert to in Grid View, 702
- inserting in Grid View, 686

External parsed entites, 902

External XSL processor, 912

F

Favorites, 920

File,

- closing, 626
- creating new, 617
- default encoding, 912
- encoding, 626
- opening, 621
- opening options, 900
- printing options, 633
- saving, 627
- sending by e-mail, 632
- tab, 900

File comparisons, 516, 517

File extensions,

- customizing, 707
- for XQuery files, 318
- setting extensions as file type, 278

File menu, 617

File paths,

- inserting in XML document, 188

File types, 902

Files,

- adding to source control, 664
- comparing two, 878
- comparison options, 883
- most recently used, 635

FileSelectionDlg,

- Application, 1150
- DialogAction, 1151
- FullName, 1151
- Parent, 1151

Find,

- and replace text in document, 647
- text in document, 644

Find in Files command, 649

Find in Files Window, 22

Find in Schemas, 292

- executing Find and Replace commands, 300
- executing Find command, 464
- global components, 302
- renaming global components, 302

replace term, 293

restricting search by property and property value, 296

restricting search to components, 295

results, 302

search term, 293

setting scope of, 299

window, 302

Find in Schemas Window, 24

Find in XBRL, 462

results, 466

search term, 462

window, 466

Find in XBRL Window, 24

Floating windows, 11

fn:base-uri in XPath 2.0,

support in Altova Engines, 1389

fn:collection in XPath 2.0,

support in Altova Engines, 1389

fn:count() in XPath 2.0,

and whitespace, 1388

fn:current-date in XPath 2.0,

support in Altova Engines, 1389

fn:current-dateTime in XPath 2.0,

support in Altova Engines, 1389

fn:current-time in XPath 2.0,

support in Altova Engines, 1389

fn:data in XPath 2.0,

support in Altova Engines, 1389

fn:deep-equal() in XPath 2.0,

and whitespace, 1388

fn:id in XPath 2.0,

support in Altova Engines, 1389

fn:idref in XPath 2.0,

support in Altova Engines, 1389

fn:index-of in XPath 2.0,

support in Altova Engines, 1389

fn:in-scope-prefixes in XPath 2.0,

support in Altova Engines, 1389

fn:last() in XPath 2.0,

and whitespace, 1388

fn:lower-case in XPath 2.0,

support in Altova Engines, 1389

fn:normalize-unicode in XPath 2.0,

support in Altova Engines, 1389

fn:position() in XPath 2.0,

and whitespace, 1388

fn:resolve-uri in XPath 2.0,

support in Altova Engines, 1389

fn:static-base-uri in XPath 2.0,
support in Altova Engines, 1389

fn:upper-case in XPath 2.0,
support in Altova Engines, 1389

Folding margin, 836

Font,
schema, 907
Schema Documentation, 907

Font size,
in Browser View, 838

Fonts in Text View, 98

Form Object Palette, 932

Form Object properties, 943

Formatting in Text View, 98

Forms,
and built-in commands, 957
and event handling, 945
and Form Objects, 943
creating new, 942
in Scripting Projects, 942
invocation of, 938
naming, 942
overview, 936
properties of, 942
setting tab sequence of objects, 943

Forward,
in Schema View, 145

Full-text,
search, 920

functions,
see under XSLT 2.0 functions, 1384
XPath 2.0 and XQuery 1.0, 1388

G

gauge chart features, 226

GCC,
compile for Linux/GCC, 560

Generate,
code from schema, 555
DB structure based on schema, 823

Generate Sample XML, 1286, 1292

GenerateSampleXMLDlg,
Application, 1162
FillWithSampleData, 1163
NonMandatoryAttributes, 1162

NonMandatoryElements, 1162
Parent, 1162
RepeatCount, 1163
TakeFirstChoice, 1163

Global,
settings, 900

Global components,
creating in Schema Overview, 115

Global declarations, 940
overview, 936

Global element,
using in XML Schema, 57

Global objects,
in SPL, 600

Global resources, 495
active configuration for, 885
changing configurations, 508
copying configurations, 502
defining, 496, 885
defining database-type, 501
defining file-type, 498
defining folder-type, 500
toolbar activation, 887
using, 504, 507, 508
using file-type and folder-type, 504

Global Resources XML File, 496

Global schema components,
finding and renaming, 302

Global scripting project,
of XMLSpy, 931

Go to File, 836

Go to line/char, 836

gradient,
background, 596

Grammar, 902

Graphics formats,
in Authentic View, 399

Gray bar, 834, 835

Grid fonts, 906

Grid view, 107, 833, 835
and data import/export, 109
and Table View, 83
appending elements and attributes, 81
data-entry in, 73
editing, 705, 706
editing in, 108
editing XML documents in (Enterprise and Professional editions), 190

Grid view, 107, 833, 835

- entry helpers in, 113
- switching to Table View, 703
- tables, 109
- using Entry Helpers, 81

Grid View Events, 948**GridView,**

- CurrentFocus, 1168
- Deselect, 1168
- IsVisible, 1168
- OnBeforeDrag, 1165
- OnBeforeDrop, 1166
- OnBeforeStartEditing, 1166
- OnEditingFinished, 1167
- OnFocusChanged, 1167
- Select, 1168
- SetFocus, 1168

GUI description, 11

H

Help,

- contents, 920
- index, 920
- key map, 921
- search, 920

Help menu, 920**Help system, 920****Hide, 917, 918****Hide markup, 169, 171****Hide markup (in Authentic View), 767****Hit Count profiling, 350****Hotkey, 888****HTML, 403**

- integration of XMLSpy, 1340

HTML documents,

- editing, 404
- Info window, 404

HTML example,

- of XMLSpyControl integration, 1331, 1332, 1333

HTML Info window, 404

I

IBM DB2,

- assigning XML Schemas to XML file, 802
- managing XML Schemas, 799
- schema management and assignment, 799

Icon,

- add to toolbar/menu, 886
- show large, 897

Identity constraint,

- toggle in Content model view, 59

Identity constraints, 135

- defining in Schema View, 115

Image formats,

- in Authentic View, 399

implementation-specific behavior,

- of XSLT 2.0 functions, 1384

implicit timezone,

- and XPath 2.0 functions, 1388

Import,

- database data as XML, 813
- database data based on XML Schema, 822
- database structure as XML Schema, 818
- MS Word document as XML, 817
- text file as XML, 811

Import/export of data,

- and Grid View, 109

Importing taxonomies, 863**Indentation,**

- in Text View, 644

Indentation guides, 836**Indentation in Text View, 98****Index,**

- help, 920

Info,

- window, 34

Info Window, 15, 917, 918

- for CSS documents, 406
- for HTML documents, 404
- in XSLT/XQuery Debugger, 339

Info window, XSLT tab,

- and creating Zip folders, 314
- and Projects, 314
- and XSLT documents, 314
- description of, 314

Info window, XSLT tab,

see also XSL Outline, 314

Information Windows,

arranging, 340

Insert,

breakpoint - XSLT debugger, 758

row (in Authentic View), 768

tracepoint - XSLT debugger, 758

Insert command,

in Grid View, 681

Installation,

location of example files, 31

Installing,

version control systems, 526

Integrating,

XMLSpy in applications, 1330

Intelligent Editing, 903**Internet, 922, 923****Internet usage,**

in Altova products, 1446

Introduction,

code generator, 556

J

JAVA,

API, 1297

ClassPath, 1297

code, 555

settings, 596

Java extension functions,

constructors, 1397

datatype conversions, Java to XPath/XQuery, 1399

datatype conversions, XPath/XQuery to Java, 1398

for XSLT and XQuery, 1392

instance methods, instance fields, 1398

overview, 1392

static methods, static fields, 1397

user-defined class files, 1393

user-defined JAR files, 1396

JRE,

for XMLSpy Plugin for Eclipse, 547

JSON, 403

convert from and to XML, 831

editing in Text View, 410

K

Key map, 921**Keyboard shortcut, 888****Key-codes,**

for your Altova software, 921

L

Language,

scripting language - changing, 938

Large markup (in Authentic View), 767**last() function,**

in XPath 1.0, 1380

last() function in XPath 2.0,

see fn:last(), 1388

Legal information, 1447**Lib,**

compiler settings, 596

Library, 609**library modules,**

in XQuery document, 1385

License, 1451

information about, 1447

License metering,

in Altova products, 1449

Licenses,

for your Altova software, 921

Line,

go to, 836

line chart features, 226**Line length,**

word wrap in text view, 835

Line margin, 836**Line numbering in Text View, 100****Line-breaks, 900****Linkbases,**

referencing, 863

Linkbases in taxonomies, 447**Linkroles in XBRL, 860****Linkroles in XBRL taxonomies, 454****Links,**

following in Authentic View, 377

Linux /GCC,

- c++ settings, 560

Little-endian, 912**loading, 1140**

M

Macro,

- add to menu/toolbar, 895
- edit button, 898

Macros,

- creating with Scripting Editor, 951
- debugging, 955
- editing with Scripting Editor, 951
- execution of, 938
- functions for, in Global Declarations, 940
- how to use in Scripting Project, 951
- overview, 936
- running, 953
- running application macros, 877
- setting as Auto-Macro in Scripting Editor, 951

Main window, 12, 34**MainCatalog, 707****MapForce, 726****Markup,**

- in Authentic View, 169, 171

Markup (in Authentic View),

- hide, 767
- show small/large/mixed, 767

Maximum cell width, 904**Memory,**

- out of exceptions, 594
- storage of schema information, 728

Memory requirements, 1441**Menu,**

- add macro to, 895
- add/delete command, 886
- Authentic, 761
- Convert, 811
- customize, 893
- Default/XMLSPY, 893
- delete commands from, 893
- DTD/Schema, 718
- Edit, 636
- Help, 920
- Project, 653

- Schema Design, 729

- SOAP, 851

- Tools, 869

- View, 833

- Window, 917

- WSDL, 840

- XML, 681

- XSL/XQuery, 748

Menu Bar, 27**Menu Browser, 838****Menus, 29****Messages Window, 16**

- in XSLT/XQuery Debugger, 339

Method,

- Reserve name, 594

Microsoft Office 2007, 181, 467**Microsoft® SharePoint® Server, 674****MIME, 902****Mixed markup (in Authentic View), 767****Mode,**

- compatible to v2005R3, 596

Mostly recently used files,

- list of, 635

Move Left command, 705**Move Right command, 705****Move up/down,**

- row (Authentic View, 769

MS Access,

- conversion of datatypes in XML Schema generation from DB, 1424
- datatypes from XML Schema, 1432

MS SQL Server,

- conversion of datatypes in XML Schema generation from DB, 1425
- datatypes from XML Schema, 1434
- schema extensions, 741
- schema settings, 742, 743

MS Visual Source Safe, 519**MSXML, 912****msxsl:script, 1405****Multi-user, 900****MySQL,**

- conversion of datatypes in XML Schema generation from DB, 1426
- datatypes from XML Schema, 1436

N

Named schema relationships,
MS SQL Server schema settings, 742

Named templates, 311

Namepsaces,
in WSDL documents, 414

Namespace,
in schemas, 38

Namespace Prefix,
inserting in Grid View, 712

namespaces,
in XBRL taxonomies, 443
in XQuery document, 1385
in XSLT 2.0 stylesheet, 1382
settings in Schema Design View, 729

Namespaces in XBRL, 861

Navigation,
shortcuts in schema design, 63

Navigation history, 145

New features, 8

New file,
creating, 617

New XML document,
creating, 69

Nillable,
not supported, 594

Node,
show curr. exec. node, 758

Non-XML files, 902

NOTATION declaration,
add as child in Grid View, 700
appending in Grid View, 694
convert to in Grid View, 703
inserting in Grid View, 687

O

OASIS,
XML catalog, 707

Object Locator,
in Database Query window, 789

Occurrences,

number of, 39

ODBC,
conversion of datatypes in XML Schema generation from
DB, 1428

ODBC Connections, 486, 779

Office Open XML, 181, 467

OOXML,
see under Office Open XML, 181, 467

Open,
file, 621

Open Office XML,
creating in Archive View, 469
editing in Archive View, 469
example files, 471

Opening options,
file, 900

Optimal Widths, 835, 904

Optional element,
making, 46

Options,
3d charts, 234
area chart features, 226
bar chart features, 226
candlestick chart features, 226
chart colors, 229
chart fonts, 236
chart grid, 231, 233, 234
chart legend, 225
chart title, 225
chart X-axis, 231
chart Y-axis, 233
chart Z-axis, 234
charts background, 225
charts sizes, 235
gauge chart features, 226
line chart features, 226
pie chart features, 226

Oracle,
conversion of datatypes in XML Schema generation from
DB, 1427
datatypes from XML Schema, 1438
schema extensions, 740
schema settings, 741

Oracle XML Db, 806
Browse database, 809
manage XML Schemas, 807

Ordering Altova software, 921

OS,

OS,

for Altova products, 1441

Out of memory, 594**Output formatting, 900****Output windows,**

toggle display on and off, 918

Overlays,

adding, 237

appending, 237

deleting, 237

for charts, 237

removing, 237

Overview, 34

of XMLSpy API, 987

P

Parameters,

in DB queries, 390

passing to stylesheet via interface, 751

Parent, 1130**Parser,**

built into Altova products, 1442

XSLT, 912

Paste,

as Text, 377

as XML, 377

Paste As,

Text, 178

XML, 178

Paste command, 637**PDF,**

transforming to in XMLSpy, 308

pie chart features, 226**Pie charts, 212****Platforms,**

for Altova products, 1441

Plug-in,

ATL sample files, 974

registration, 969

User interface configuration, 971

XMLSPY, 968

Position,

Character, 836

Line, 836

position() function,

in XPath 1.0, 1380

position() function in XPath 2.0,

see fn:position(), 1388

PowerPoint 2007, 181, 467**Presentation, 904****Pretty-print,**

in Text View, 644

Print setup, 634**Printing,**

from Authentic View, 375

Printing options, 633**Private key of certificates, 257****Processing Instruction,**

add as child in Grid View, 697

appending in Grid View, 690

convert to in Grid View, 702

inserting in Grid View, 683

Profiler, 350, 755**Profiling,**

Callgraph, 350

Hit Count, 350

Program settings, 900**Programmers' Reference, 926****Programming points,**

in Scripting Project, 956

Project,

properties, 678

window, 34

Project management in XMLSpy, 94**Project menu, 653****Project Window, 13, 917, 918**

toggle display on and off, 918

Projects,

adding active files to, 671

adding external folders to, 671

adding external Web folders to, 674

adding files to, 670

adding folders to, 671

adding global resources to, 670

adding related files to, 671

adding to source control, 664

adding URL to, 670

batch processing with, 514

benefits of using, 514

closing, 656

creating new, 655

how to create and edit, 510

location of installed files, 31

Projects,

- most recently used, 680
- naming, 510
- opening, 655
- overview, 653
- overview of, 509
- properties of, 510
- reloading, 656
- saving, 510, 656
- using, 514

Projects in XMLSpy,

- benefits of, 94
- how to create, 94

Properties and Events pane, 932**Provider,**

- source control, 519

PUBLIC,

- identifier - catalog, 707

Public key of certificates, 257**PVCS Version Manager, 519**

Q

QName serialization,

- when returned by XPath 2.0 functions, 1389

Queries,

- for DB display in Authentic View, 390

Query,

- see under Database Query window, 786
- see under Query Database, 786
- see under XQuery, 786

Query Database command, 786**Query pane,**

- in Database Query window, 793

R

Redo command, 636**Regions in SQL scripts, 793****Register,**

- plug-in, 969

Registering your Altova software, 921**Registry,**

- settings, 900

Regular expressions, 649

- in search string, 644

Relationships in Taxonomies, 454, 456, 459**Reload, 900****Reloading,**

- changed files, 626

Remove,

- breakpoint - XSLT debugger, 758
- tracepoint - XSLT debugger, 758

Removing,

- overlay charts, 237

Repeated elements, 903**Replace, 22**

- text, 644
- text in document, 647
- text in multiple files, 649

Repositories, 519**Reserve,**

- method name, 594

Reset,

- menu commands, 893
- shortcut, 888
- toolbar & menu commands, 887

Resources,

- increase - background status updates, 519

Restart,

- XSLT debugger, 757

Return key,

- see Enter key, 400

RichEdit, 768**RichEdit 3.0, 909****Right-to-left writing systems, 1445****Row,**

- append (in Authentic View), 768
- delete (in Authentic View), 768
- duplicate (in Authentic View), 768
- insert (in Authentic View), 768
- move up/down, 769

Row in Grid View,

- appending, 704
- inserting, 704

Rules,

- for schema validation (see Schema Rules), 271

S

save, 1132

Saving files,

encoding of, 626

schema, 1037

also see XML Schema, 718

assigning to DB XML, 802

code generator, 555

converting to UML, 724

create DB based on schema, 823

Design view, 833

documentation, 65

Documentation font, 907

management and assignment in IBM DB2 databases, 799

open WSDL schema, 419

see XML Schema, 35

settings, 900

Schema Design menu, 729

Schema Design View,

Display all globals, 740

Display diagram, 740

zoom feature, 739

Schema editing,

annotations, 115, 120

comments, 115, 120

content model, 120

processing instructions, 115, 120

Schema fonts, 907

Schema Overview, 35, 115

Schema Rules, 271

adding Rule Sets to a schema, 271

defining, 273

Schema Subsets, 267, 745, 746

schema validation of XML document,

for XQuery, 1385

Schema View, 115

configuring the view, 44

entry helpers, 132

moving back and forward, 145

Schema View, searching in,

see Find in Schemas, 292

SchemaAgent,

connect to server from XMLSpy, 743

disconnect from server, 744

display schemas in, 744

extended validation, 745

opening schemas from XMLSpy, 290

working with, 285, 286, 290

SchemaAgent in XMLSpy, 282

SchemaAgent Server,

connecting to, 283

schema-awareness,

of XPath 2.0 and XQuery Engines, 1388

SchemaDocumentationDlg,

AllDetails, 1170

Application, 1170

IncludeAll, 1172

IncludeAttributeGroups, 1172

IncludeComplexTypes, 1172

IncludeGlobalElements, 1173

IncludeGroups, 1173

IncludeIndex, 1173

IncludeLocalElements, 1174

IncludeRedefines, 1174

IncludeSimpleTypes, 1175

OptionsDialogAction, 1175

OutputFile, 1176

OutputFileDialogAction, 1176

OutputFormat, 1176

Parent, 1177

ShowAnnotations, 1177

ShowAttributes, 1177

ShowChildren, 1177

ShowConstraints, 1178

ShowDiagram, 1178

ShowEnumerations, 1178

ShowNamespace, 1179

ShowPatterns, 1179

ShowProgressBar, 1179

ShowProperties, 1179

ShowResult, 1180

ShowSingleFacets, 1180

ShowSourceCode, 1180

ShowType, 1181

ShowUsedBy, 1181

schemanativetype, 598

Schemas,

in memory, 728

managing for IBM DB2, 799

Schemas, finding in,

see Find in Schemas, 292

Script language, 914

Scripting, 914**Scripting Editor,**

- GUI description, 932

- Main Window, 932

- starting, 877

Scripting Environment, 928

- usage overview, 930

Scripting language, 938**Scripting Project,**

- and Events, 948

- application event handlers, 948

- Event Handlers, 936

- Forms, 936

- Forms in, 942

- Global Declarations, 936

- Global Declarations in, 940

- Macros, 936

- Macros in, 951

- programming points, 956

- steps for creating, 938

Scripting Project Tree pane, 932**Scripting Projects,**

- for XMLSpy, 931

- for XMLSpy Project, 931

Scripts in XSLT/XQuery,

- see under Extension functions, 1392

Search,

- help, 920

- see Find, 647

Searching in schemas,

- see Find in Schemas, 292

Searching in XBRL,

- see Find in XBRL, 462

Select All command, 644**Sequence compositor,**

- using, 39

Settings, 29, 900

- 3d charts, 234

- area chart features, 226

- bar chart features, 226

- c++ and c#, 596

- candlestick chart features, 226

- chart background, 225

- chart colors, 229

- chart fonts, 236

- chart grid, 231, 233, 234

- chart legend, 225

- chart title, 225

- chart X-axis, 231

- chart Y-axis, 233

- chart Z-axis, 234

- charts sizes, 235

- gauge chart features, 226

- line chart features, 226

- pie chart features, 226

- scripting, 914

- XSLT Debugger, 760

Settings for file comparison, 883**SharePoint® Server, 674****Shortcut, 888**

- assigning/deleting, 888

- show in tooltip, 897

Show, 917, 918**Show curr. exec. nodes,**

- XSLT debugger, 758

Show large markup, 169, 171**Show mixed markup, 169, 171****Show small markup, 171****Show small markup, 169****Side-by-side, 904****Signature,**

- see XML Signature, 712

Signatures,

- see XML signatures, 250

Simple type,

- in schema definitions, 50

Size, 909**Small markup (in Authentic View), 767****Smart Restrictions, 139****SOAP, 413, 424**

- create new request, 851

- debugger options, 857

- debugger session, 855

- request, 857

- request parameters, 854

- requests, 853, 856

- send request to server, 853

- sending request from WSDL, 420

- start proxy server, 856

- stop proxy server, 857

SOAP communication process, 425**SOAP Debugger,**

- in Visual Studio .NET, 544

- setting breakpoints, 432

- usage example, 426

SOAP menu, 851

SOAP validation, 424**Software product license, 1451****Source control, 519, 915**

- add to source control, 664
- changing provider, 669
- checking in, 662
- checking out, 661
- enabling, disabling, 658
- get latest version, 659
- getting files, 659
- getting folders, 660
- open project, 657
- properties, 668
- refresh status, 669
- removing from, 664
- sharing from, 665
- show differences, 667
- show history, 666
- supported providers, 656
- undo check out, 663

Source control manager, 669**Source folding in Text View, 100****Speed up,**

- increase - background status updates, 519

Spelling checker, 869

- custom dictionary, 869

Spelling options, 873**SPL, 597**

- code blocks, 597
- conditions, 602
- foreach, 603
- global objects, 600
- subroutines, 604
- using files, 601
- variables, 599

Splash screen, 904**SPP file locations, 653****SPS,**

- assigning to new XML file, 617

SPS file,

- assigning to XML file, 763

SPS tables,

- editing dynamic tables, 169

SPS tables in Authentic View,

- usage of, 382

SpyProject,

- CloseProject, 1182
- ProjectFile, 1182

RootItems, 1183

SaveProject, 1183

SaveProjectAs, 1183

SpyProjectItem,

- ChildItems, 1184
- FileExtensions, 1184
- ItemType, 1184
- Name, 1184
- Open, 1184
- ParentItem, 1184
- Path, 1185
- ValidateWith, 1185
- XMLForXSLTransformation, 1185
- XSLForXMLTransformation, 1185
- XSLTransformationFileExtension, 1185
- XSLTransformationFolder, 1186

SpyProjectItems,

- AddFile, 1186
- AddFolder, 1186
- AddURL, 1187
- Count, 1187
- Item, 1187
- RemoveItem, 1187

SQL Editor,

- creating query in, 796
- description of, 793
- in Database Query window, 793

SQL Server,

- manage XML Schemas, 804

Start,

- XSLT debugger, 756

Start group,

- add (context menu), 898

StarTeam, 519**Static (SPS) tables in Authentic View,**

- usage of, 382

Status,

- background updates, 533

Status Bar, 27**Step into,**

- XSLT debugger, 757

Step out,

- XSLT debugger, 757

Step over,

- XSLT debugger, 757

Stop,

- XSLT debugger, 757

Structured text, 903

Style, 906, 909

Stylesheet PI, 755, 756

StyleVision, 726

and XBRL, 868

for editing StyleVision Power Stylesheet, 763

StyleVision Power Stylesheet,

assigning to XML file, 763

editing in StyleVision, 763

Support Center, 922

Support options, 31

Sybase,

conversion of datatypes in XML Schema generation from DB, 1430

Syntax checking of JSON documents, 410

Syntax coloring,

for XQuery, 319

Syntax-coloring, 902, 904

T

Tab characters, 900

Tab size,

and pretty-printing, 836

setting, 836

Table,

build automatically, 902

colors, 910

Table command,

in Grid View, 703

Table of contents, 920

Table View, 903

and switching to Grid View, 703

append row, 704

how to use, 83

insert row, 704

sorting columns, 704, 705

Tables,

editing dynamic (SPS) tables, 169

in Authentic View, 369

in Grid View, 109

Tables in Authentic View,

icons for editing XML tables, 387

usage of, 381

using SPS (static and dynamic) tables, 382

using XML tables, 383

Target namespaces in XBRL, 862

Taxonomies,

adding elements to, 450

and linkbases, 447

and linkroles, 454

and New Taxonomy Wizard, 439

creating new, 439

files in, 437, 447

importing, 445, 863

namespaces in, 443, 861

relationships in, 454, 456, 459

steps for creating, 436

target namespace of, 443

target namespaces in, 862

Taxonomies in XBRL, 436, 437

Technical Information, 1440

Technical Support, 922

Template files,

for new documents, 617

Template folder, 33

Template XML File,

in Authentic View, 366

Templates,

of XML documents in Authentic View, 761

Templates Window,

in XSLT/XQuery Debugger, 339

terminate, 1041

Text,

add as child in Grid View, 696

appending in Grid View, 689

convert to in Grid View, 701

editing in Authentic View, 377

find and replace, 647

finding in document, 644

font, 909

formatting in Authentic View, 377

inserting in Grid View, 682

pretty-printing, 644

Text file,

export of XML data to, 826

import as XML, 811

Text view, 98, 833

and commenting in XML documents, 188

and empty lines in XML documents, 188

auto-complete enable/disable, 903

bookmarks in, 100

editing in, 73

Entry helpers in, 105

font properties, 98

Text view, 98, 833

- formatting of text, 98
- indentation, 98
- indentation in, 100
- intelligent editing features, 103
- line numbering in, 100
- schema fonts, 907
- source folding in, 100
- special editing features for XML documents, 188
- word-wrapping, 98

Text View Events, 948**Text View Settings dialog, 836****TextImportExportSettings,**

- DestinationFolder, 1188
- EnclosingCharacter, 1188
- Encoding, 1189
- EncodingByteOrder, 1189
- FieldDelimiter, 1189
- FileExtension, 1189
- HeaderRow, 1189
- ImportFile, 1190

Tile,

- horizontally, 917
- vertically, 917

Toggle, 917, 918**Toolbar, 27**

- activate/deactivate, 887
- add command to, 886
- add macro to, 895
- create new, 887
- reset toolbar & menu commands, 887
- show large icons, 897

Tools menu, 869**Tooltip,**

- show, 897
- show shortcuts in, 897

Topic,

- view on TOC, 920

Trace window, 344

- in XSLT/XQuery Debugger, 340

Tracepoint,

- dialog box, 759

Tracepoints,

- using in XSLT/XQuery Debugger, 344

Transformation,

- see XSLT transformation, 750

Trusted locations for Authentic scripts, 769**Turn off automatic validation, 902****Tutorial,**

- example files, 33
- for WSDL, 414
- goals, 33

Tutorials,

- location of installed files, 31

type,

- extension in XML document, 71

Types,

- built in, 608

U

UCS-2, 912**UML,**

- converting schemas to, 724

Undo command, 636**Unicode,**

- support in Altova products, 1444

Unicode support,

- in Altova products, 1444, 1445

Unnamed element relationships,

- MS SQL Server schema settings, 743

Unselected, 835**Update,**

- background status updates, 533

Update Entry Helpers command, 711**URL, 1043, 1044, 1132, 1140, 1141**

- sending by e-mail, 632

User interface,

- configure using plug-in, 971

User interface description, 11**User Manual, 3, 6****User Reference, 616****UTF-16, 912**

V

Validate,

- WSDL file, 419

Validating,

- XML documents, 77

Validating XML documents, 186**Validation, 29, 707**

Validation, 29, 707

- assigning DTD to XML document, 718
- assigning XML Schema to XML document, 719
- extending with Schema Rules, 271
- of related schemas using SchemaAgent, 290
- WSDL files, 710

Validation messages, 16**Validator,**

- in Altova products, 1442

Variables,

- in SPL, 599

Variables Window,

- in XSLT/XQuery Debugger, 337

Version control,

- Diffdog differencing editor, 533
- installation procedures, 526

Version Number, 923**View,**

- Browser view, 834
- Collapse, 835
- Enhance Grid view, 833
- Expand, 834, 835
- Go to File, 836
- Go to line/char, 836
- Optimal widths, 835
- Schema Design view, 833
- Text View, 833

View menu, 833**Visual Studio .Net,**

- and XMLSpy, 539
- and XMLSpy Debuggers, 544
- and XMLSpy differences, 542
- know issues wrt XMLSpy, 545

VS .NET,

- and XMLSpy Integration Package, 540

W

Watch for changes, 900**Web Server, 922, 923****web service,**

- connecting to, 419

Well-formed test of JSON documents, 410**Well-formedness check, 706**

- for XML document, 77

Well-formedness of XML documents, 186**whitespace handling,**

- and XPath 2.0 functions, 1388

whitespace in XML document,

- handling by Altova XSLT 2.0 Engine, 1382

Whitespace markers, 836**whitespace nodes in XML document,**

- and handling by XSLT 1.0 Engine, 1380

Window,

- Cascade, 917
- Entry-Helper, 918
- Info, 917, 918
- Open, 918
- Project, 917, 918
- Tile horizontally, 917
- Tile vertically, 917

Window menu, 917**Windows,**

- auto-hiding, 11
- floating, docking, tabbing, 11
- managing display of, 11
- overview, 34
- support for Altova products, 1441

Wizard for new taxonomies, 439**Word 2007, 181, 467****Word document,**

- import as XML, 817

Word wrap,

- enable/disable, 835

Word-wrapping in Text View, 98**Wrap,**

- word wrap enable/disable, 835

WSDL, 413

- 1.1 components, 840
- 2.0 components, 842
- binding in 1.1, 841
- binding in 2.0, 844
- connecting to a web service, 419
- converting from 1.1 to 2.0, 849
- create documentation, 421
- create new document, 414
- creating bindings, 416
- creating messages, 415
- creating operations, 415
- creating parameters, 415
- creating ports, 418
- creating PortTypes, 415
- creating services, 418
- generate documentation, 845

WSDL, 413

- interface in 2.0, 842
- messages in 1.1, 840
- namespaces, 414
- open schema, 419
- operations in 1.1, 841
- portType in 1.1, 841
- reparse document, 849
- sending SOAP request, 420
- service in 1.1, 842
- service in 2.0, 844
- SOAP debugger example, 426
- types, 845
- validating, 419
- web service, 851

WSDL Design View,

- Bindings, 148
- description of, 147
- file viewing in, 147
- functionality, 147
- Main Window, 148
- PortTypes, 148
- Services, 148

WSDL files,

- Validation, 710

WSDL fonts, 907**WSDL menu, 840****WSDL tutorial, 414****WSDL View,**

- Details entry helper, 152
- entry helpers, 152
- importing into WSDL document, 152
- Overview entry helper, 152

X

XBRL, 435

- and MapForce, 868
- and StyleVision, 868
- arcroles, 858
- generate documentation, 864
- linkroles, 860
- namespaces, 861
- target namespaces, 862

XBRL fonts, 908**XBRL menu, 858****XBRL taxonomies, 436**

- see also Taxonomies, 437

XBRL View, 157

- Calculation tab in main window, 161
- Definition tab in main window, 161
- Elements tab in main window, 157
- entry helpers in, 164
- Presentation tab in main window, 161
- settings, 867

XBRL View, searching in,

- see Find in XBRL, 462

Xerces,

- support, 596

XInclude, 640, 683, 690, 697

- adding as child in Grid View, 697
- appending in Grid View, 690
- inserting in Grid View, 640, 683
- inserting in Text View, 640
- inserting in XML document, 188

XML,

- copying as structured text, 638
- Oasis catalog, 707
- spelling checker, 869

XML data,

- exporting to database, 829
- exporting to text file, 826

XML DB,

- loading new data row into Authentic View, 764
- loading new XML data row, 389

XML Diff,

- comparing directories, 880
- comparing files, 878, 883

XML document,

- assigning to XSLT stylesheet, 756
- browse Oracle XML Db, 809
- creating new, 69
- editing in Text View, 73
- generating from DTD, 264
- generating from XML Schema (Enterprise and Professional editions), 266
- opening in Authentic View, 366

XML document creation,

- tutorial, 69

XML documents, 183

- and commenting in Text View, 188
- and empty lines in Text View, 188
- and XPath expression of a node, 188
- and XQuery, 197

XML documents, 183

- assigning schemas (incl. DTDs), 186
- automatic validation, 184
- automating XQuery executions of, 197
- automating XSLT transformations of, 197
- checking validity of, 77
- checking well-formedness, 186
- default views of, 184
- editing in Authentic View, 193
- editing in Grid View (Enterprise and Professional editions), 190
- encoding of, 261
- evaluating XPath expressions on, 261
- generating schemas from, 261
- importing and exporting text, 261
- inserting file paths in, 188
- inserting XInclude, 188
- opening, 184
- saving, 184
- searching and replacing in, 261
- Text View editing features for, 188
- transforming with XSLT, 197
- validating, 186

XML file,

- generate from DTD or XML Schema, 726

XML Import,

- based on schema, 822

XML menu, 681**XML Parser,**

- about, 1442

XML prolog,

- add as child in Grid View, 696
- appending in Grid View, 690
- convert to in Grid View, 702
- inserting in Grid View, 683

XML Schema,

- adding components, 39
- adding elements with, 43
- also see Schema, 718
- assigning to DB XML, 802
- assigning to XML document, 719
- configuring Content Model View, 736
- configuring the view, 44
- content model diagram, 731
- converting to DTD, 722
- creating a basic schema, 35
- creating a new file, 35
- defining namespaces in, 38

- generate outline XML file from, 726
- generating code from, 727
- generating documentation of, 731
- generating from DTD (Enterprise and Professional editions), 264
- generating from XML document, 720
- go to definition in from XML document, 720
- go to from XML document, 719
- management and assignment in IBM DB2 databases, 799
- managing for IBM DB2, 799
- menu commands related to, 718
- modifying while editing XML document, 87
- MS SQL Server extensions, 741
- MS SQL Server schema settings, 742, 743
- namespaces settings in Schema Design View, 729
- navigation in design view, 63
- Oracle extensions, 740
- Oracle schema settings, 741
- settings in Schema Design View, 729
- tutorial, 35

XML Schema datatypes,

- converted to MS Access datatypes, 1432
- converted to MS SQL Server datatypes, 1434
- converted to MySQL datatypes, 1436
- converted to Oracle datatypes, 1438
- in generation of XML Schema from ADO DB, 1429
- in generation of XML Schema from MS Access DB, 1424
- in generation of XML Schema from MS SQL Server DB, 1425
- in generation of XML Schema from MySQL DB, 1426
- in generation of XML Schema from ODBC DB, 1428
- in generation of XML Schema from Oracle DB, 1427
- in generation of XML Schema from Sybase DB, 1430

XML schema definitions,

- advanced, 50

XML Schemas, 263

- and global resources, 186
- converting to DTD (Enterprise and Professional editions), 266
- editing in Grid View (Enterprise and Professional editions), 266
- editing in Schema View (Enterprise and Professional editions), 266
- editing in Text View, 266
- generating XML document from, 266
- plus DTDs, 186

XML Signature, 765**XML Signatures, 250, 398**

- creating, 252, 712

XML Signatures, 250, 398

verifying, 255, 715

XML tables in Authentic View,

icons for editing, 387

usage of, 383

XML text,

copying, 637

xml:base, 144

and XInclude, 640, 683, 690, 697

xml:id, 144**xml:lang, 144****xml:space, 144****XML-Conformance, 902****XMLData,**

AppendChild, 1228

EraseAllChildren, 1229

EraseCurrentChild, 1230

GetChild, 1230

GetChildKind, 1231

GetCurrentChild, 1232

GetFirstChild, 1232

GetNextChild, 1233

HasChildren, 1234

HasChildrenKind, 1234

InsertChild, 1234

IsSameNode, 1236

Kind, 1236

MayHaveChildren, 1236

Name, 1236

Parent, 1237

TextValue, 1237

XMLSchemas,

flattening included schemas, 267, 746

including other schemas in, 267

splitting into subsets, 745

XMLSpy, 616, 923

features, 31

help, 31

integration, 1330

plug-in registration, 969

XMLSpy API,

accessing, 1357

documentation, 985

overview, 987

XMLSpy command table, 1345**XMLSpy Debugger perspectives in Eclipse, 554****XMLSpy Enterprise Edition,**

user manual, 3

XMLSpy in Eclipse, 546**XMLSpy integration,**

example of, 1331, 1332, 1333

XMLSpy Integration Package, 540, 547**XMLSpy perspective in Eclipse, 551****XMLSPY plug-in, 968****XMLSpy Plugin for Eclipse,**

installing, 547

XMLSpy Plugin for VS .NET,

installing, 540

XMLSpyCommand,

in XMLSpyControl, 1358

XMLSpyCommands,

in XMLSpyControl, 1359

XMLSpyControl, 1360

documentation of, 1330

example of integration at application level, 1331, 1332, 1333

examples of integration at document level, 1335

integration at application level, 1331

integration at document level, 1334, 1335

integration using C#, 1335, 1336

integration using HTML, 1340

object reference, 1358

XMLSpyControlDocument, 1366**XMLSpyControlPlaceHolder, 1372****XMLSpyDocumentEditor,**

MarkUpView, 1276

XMLSpyLib, 985, 987

Application, 1027

AuthenticDataTransfer, 1046

AuthenticRange, 1050

AuthenticSelection (obsolete), 1254

AuthenticView, 1077

CodeGeneratorDlg, 1094

DatabaseConnection, 1100

Dialogs, 1106

DocEditEvent (obsolete), 1238

DocEditView (obsolete), 1257

Document, 1109

Documents, 1137

DTDSchemaGeneratorDlg, 1141

ElementList, 1145

ElementListItem, 1146

ExportSettings, 1147

FileSelectionDlg, 1150

GenerateSampleXMLDlg, 1161

GridView, 1165

XMLSpyLib, 985, 987

- ProjectItem, 1183
- SchemaDocumentationDlg, 1169
- SpyProject, 1182
- SpyProjectItems, 1186
- TextImportExportSettings, 1188
- XMLData, 1227

XML-Text, 903**XPath,**

- evaluating, 17
- generating of a node in an XML document, 188

XPath 1.0,

- in XPath Evaluator, 706

XPath 2.0,

- in XPath Evaluator, 706

XPath 2.0 functions,

- general information about, 1388
- implementation information, 1388
- see under fn: for specific functions, 1388

XPath Evaluator,

- usage, 706

XPath functions support,

- see under fn: for individual functions, 1389

XPath of selected node in XML document,

- copying to the clipborad, 640

XPath to selected node, 168**XPath Window, 17****XPaths,**

- setting for tracepoints, 344

XPath-Watch Window,

- in XSLT/XQuery Debugger, 338

XPointer,

- generating of a node in an XML document, 188

XPointer of selected node in XML document,

- copying to the clipborad, 640

XPointers, 640, 683, 690, 697**XQuery,**

- DB support, 323
- document validation, 322
- editing in Text View, 317
- engine information, 1379
- entry helpers, 319
- execution, 322
- Extension functions, 1392
- for querying XML databases, 323
- functions for IBM DB2, 323
- intelligent editing features, 321
- opening file, 318

- passing variables to the XQuery document, 751

- syntax coloring, 319

XQuery 1.0 Engine,

- information about, 1385

XQuery 1.0 functions,

- general information about, 1388
- implementation information, 1388
- see under fn: for specific functions, 1388

XQuery Debugger,

- see XSLT/XQuery Debugger, 327

XQuery documents,

- analyzing execution time of, 350, 755

XQuery Execution, 755**XQuery files,**

- setting file extensions in XMLSpy, 318

XQuery processor,

- in Altova products, 1443

XQuery Profiler,

- charts of results, 360

xs:QName,

- also see QName, 1389

xsi:type,

- usage, 71

XSL,

- see XSLT, 756

XSL Outline, 310**XSL Outline window, 22, 311****XSL transformation,**

- see XSLT, 90

XSL/XQuery menu, 748**xsl:call-template, 311****XSL:FO,**

- and XSLT transformations, 308

xsl:param, 311**xsl:preserve-space, 1380****xsl:strip-space, 1380****xsl:with-param, 311****XSLT, 836**

- and batch transformations, 308
- auto-completion in Text View, 306
- documents, 306
- engine information, 1379
- entry helpers for, 306
- Extension functions, 1392
- functionality in XMLSpy, 306
- modifying in XMLSpy, 92
- processor, 912
- transformations in XMLSpy, 308

XSLT, 836

validating, 306

XSLT 1.0 Engine,

limitations and implementation-specific behavior, 1380

XSLT 2.0 Engine,

general information about, 1382

information about, 1382

XSLT 2.0 functions,

implementation-specific behavior of, 1384

see under fn: for specific functions, 1384

XSLT 2.0 stylesheet,

namespace declarations in, 1382

XSLT Debugger,

breakpoints/tracepoints dialog box, 759

debug windows, 760

enable/disable breakpoint, 758

enable/disable tracepoint, 759

end debugger session, 757

in Visual Studio .NET, 544

insert/remove breakpoint, 758

insert/remove tracepoint, 758

restart debugger, 757

settings, 760

show curr. exec. nodes, 758

start debugger, 756

step into, 757

step out, 757

step over, 757

stop debugger, 757

XSLT document structure, 22**XSLT documents,**

and Info window, 314

editing and managing with XSL Outline, 310

XSLT functions,

in XSL Outline window, 311

XSLT parameters,

passing to stylesheet via interface, 751

XSLT processors,

in Altova products, 1443

XSLT Profiler,

charts of results, 360

XSLT stylesheet,

assigning to XML document, 755

assigning XML document to, 756

opening, 756

XSLT stylesheet for FO,

assigning to XML document, 756

XSLT stylesheets,

analyzing execution time of, 350, 755

XSLT templates,

in XSL Outline window, 311

XSLT transformation, 749

assigning XSLT file, 90

in XMLSpy, 91

to FO, 750

to PDF, 750

tutorial, 90

XSLT/XQuery Debugger,

breakpoints usage, 341

Call Stack Window, 338

Context window, 337

description of interface, 328

description of mechanism, 328

features and usage, 327

Info Window, 339

information windows, 336

Messages Window, 339

settings, 332

starting a session, 334

Templates Window, 339

toolbar icons, 329

Trace Window, 340

tracepoints usage, 344

Variables Window, 337

XPath-Watch Window, 338

XSLT/XQuery debugging,

files used, 327

XSLT/XQuery Profiler, 350

Z

ZIP files, 181, 467

creating in Archive View, 472

editing in Archive View, 472

Zoom feature,

in Schema Design View, 739

Zooming in Text View, 100