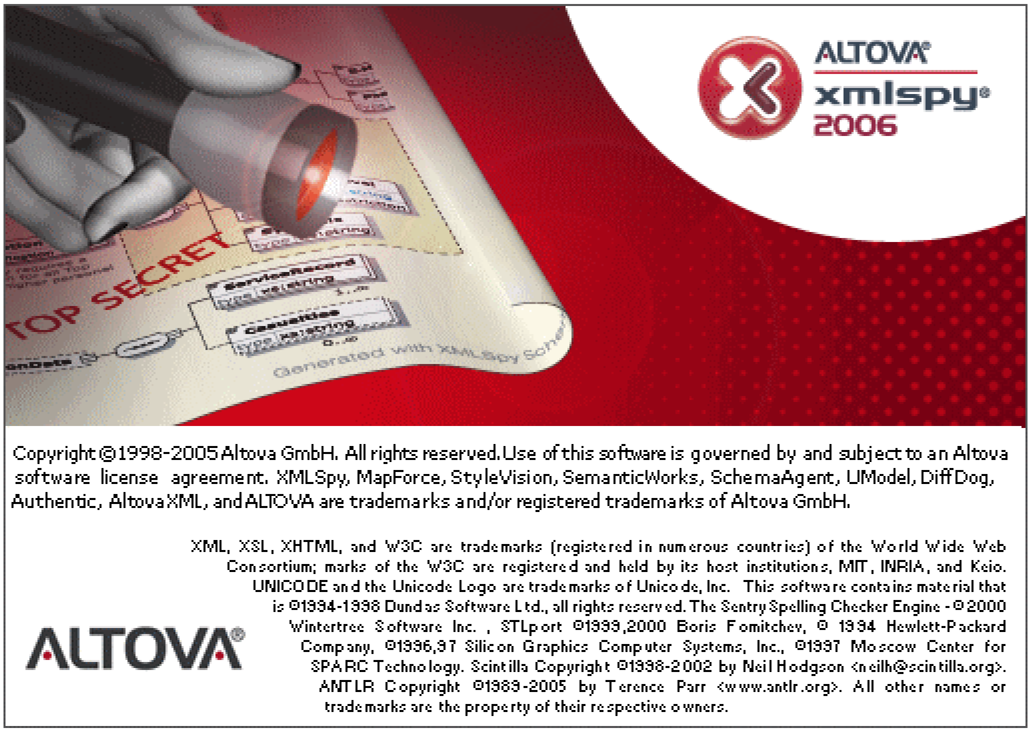
User Manual and Programmers' Reference



Altova XMLSpy Professional Edition User Manual

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: 2006

Table of Contents

Appendices 754

1 Engine Information 755

2 Datatypes in DB-Generated XML Schemas 768

3 Datatypes in DBs Generated from XML Schemas 776

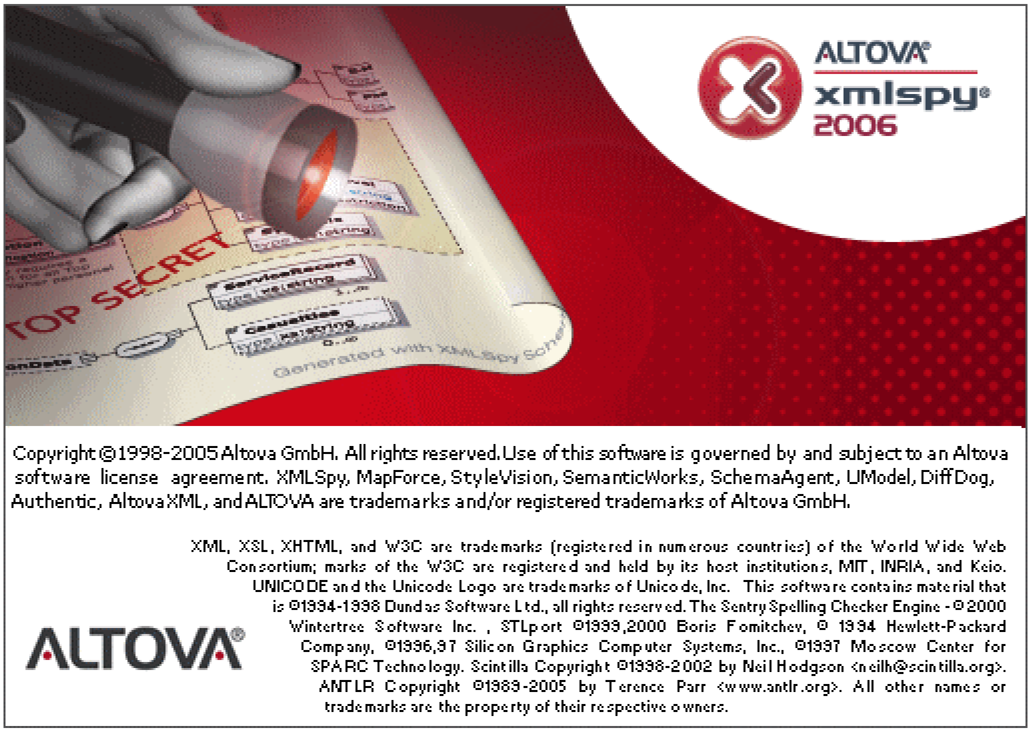
4 Technical Data 785

Altova XMLSpy Professional Edition User Manual

Welcome to XMLSpy Professional Edition

Welcome to XMLSpy Professional Edition

Altova® XMLSpy® 2006 Professional Edition is the industry standard XML Development Environment for designing, editing and debugging enterprise-class applications involving XML, XML Schema, XSL/XSLT, and XQuery technologies. It is the ultimate productivity enhancer for J2EE, .NET and database developers.



Altova XMLSpy Professional Edition User Manual

User Manual

User Manual

This User Manual contains tutorials of the various XMLSpy features to help you get started, and explanations of important features. It also contains a comprehensive reference section that describes XMLSpy IDE features. The User Manual consists of the following sections:

- An Introduction, which provides an overview of Altova's XML products.
- A **Tutorial** that shows you how to use XMLSpy for the major aspects of XML:
  - XML editing and validation
  - Schema/DTD editing and validation
  - XSLT editing and transformation
- Detailed descriptions of the five XMLSpy views: Text View, Enhanced Grid View, Schema/WSDL View, Authentic View, and Browser View. These five sections describe the GUI and features of the various views, including descriptions of the various components, symbols, and icons found in each view. In these sections, you will become familiar with the views and learn how to use each view's features.
- A description of how XMLSpy can be integrated with **Visual Studio .NET**.
- A description of **Eclipse** integration in XMLSpy.
- A description of how to use the **XSLT/XQuery Debugger** component, which is delivered with XMLSpy. The XSLT/XQuery Debugger enables you to check and correct your XSLT 1.0 and XSLT 2.0 stylesheets and XQuery documents. In an XSLT debugging session, you can go through the transformation of an XML file, step-by-step, using a selected stylesheet. In an XQuery debugging session, you can step through the execution of an XQuery document.
- A **User Reference** that contains a description of all menu commands available in XMLSpy.

1 Introduction

This introduction briefly describes:

- Altova's various XML products and how they relate to each other
- The main features of XMLSpy
- The user interface

This section is intended to serve as a general introduction to XMLSpy, and will familiarize you with the product's capabilities and interface.

1.1 Altova's XML Products

Altova's XML products are easy to use and cover all your XML needs. They complement each other and provide you with a comprehensive XML application development environment. The Altova family currently comprises the following products.

Altova XMLSpy® 2006 is a comprehensive IDE for developing XML projects, and is available in three feature configurations. At the top end, the Enterprise Edition provides an efficient and flexible environment for creating and editing DTDs, XML Schemas, XML files, and XSLT stylesheets. It has powerful editing features, multiple document views—including Altova's own Authentic View—validation, and XSLT transformations with an internal processor. It can import, and export to, text files and databases. Among other features are management of XML documents in projects, an XSLT and XQuery Debugger, a WSDL Editor, and code generation. The Professional and Home Editions have different feature configurations.

Altova StyleVision® 2006 is a new approach to writing complex XSLT stylesheets using an intuitive drag-and-drop user interface. With StyleVision you also create StyleVision Power Stylesheets that are used to control the display and data entry of XML documents in Authentic View. StyleVision is available in Enterprise and Professional editions.

Altova Authentic® 2006 (Desktop and Browser editions) are word-processor type editor programs which support form-based data input of XML documents. You can insert components such as graphics and tables, and validate in real-time against a schema.

Altova MapForce® 2006 is a product for mapping one schema to another and converting XML files based on one schema to XML files based on another.

Altova website

You may also want to periodically check the Altova website, www.altova.com, for news, updates, additional documentation, and support.

Please note that user manuals for all Altova products are available in the following formats:

- Online 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

The documentation on the website is updated periodically and kept up-to-date with the current versions.

Support and feedback

If you require additional information or have a query about Altova products, please don't hesitate to 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 will be processed by our support team

Also, please feel free to send us your feedback about the documentation at http://www.altova.com/support_center.html.

1.2 XMLSpy's main features

XMLSpy is an integrated Development Environment (IDE) for the development of XML projects. XMLSpy can be used, among other things, to edit and process a variety of XML and other text documents; to import to and export from XML documents (including to and from databases); to convert between certain types of XML documents and other document types; to link different types of XML documents in projects; process documents with the built-in XSLT 1.0 processor, XSLT 2.0 processor and XQuery 1.0 processor, and to even generate code from XML documents.

XMLSpy also provides a graphical editing view of XML documents in Altova's popular Authentic View, thus enabling users to enter data into an XML document as they would into a wordprocessor-type application. Authentic View is particularly useful in situations where:

- people not familiar to XML are called upon to enter data into an XML document, or where
- several users input data into, or view, a single document located on a server or shared resource.

In this section, we provide a brief overview of the **main features** of XMLSpy. These features are described in more detail in the various interface view sections (Text View, Schema/WSDL View, Authentic View, etc) of this document and in the User Reference. Please note that this is not a comprehensive list of available features. It is intended to give you a broad idea of what is possible with XMLSpy.

Edit XML documents in multiple editing formats
You can edit an XML document as plain text (Text View), in a hierarchical table format (Enhanced Grid View), or in a graphical WYSIWYG view (Authentic View). For XML Schemas, you can also use Schema/WSDL View, which is a graphical interface that greatly simplifies the creation of complex schemas. You can also switch among the various views to suit your convenience. The Browser View enables you to directly view XML documents associated with an XSLT stylesheet and HTML documents.

Well-formedness checking and built-in validator
All XML documents are checked for well-formedness when you change views or save the file. XML documents can also be validated if a schema (DTD or XML Schema) is associated with the XML document. Other types of documents, such as DTDs, are also checked for errors in syntax and structure.

Structural editing features
In Text View, features such as line-numbering, indentation, bookmarks, and expandable and collapsible elements help you to navigate your document quickly and efficiently.

Intelligent Editing
If a schema is associated with an XML document, the auto-completion feature of Text View provides valuable editing help. As you type, pop-up menus containing the elements, attributes, and enumerated attribute values allowed at the cursor point appear. Additionally, the correct closing tags are automatically inserted when you complete opening tags, and attributes selected from the pop-up menu are inserted with opening and closing quotes. You can also enable XMLSpy to automatically insert mandatory elements and/or attributes when an element is inserted. Further, each view has a set of Entry Helpers that enable you to insert document components or specify the properties of the component selected in the Main Window.

Schema editing and management

You can create XML Schemas quickly and easily in the graphical Schema/WSDL View. This takes away much of the difficulty of knowing XML Schema structures, syntax, and design principles. You can also create DTDs that can be checked for correct syntax, plus convert between Schemas and DTDs, and generate documentation.

Built-in XSLT 1.0 and XSLT 2.0 processors

The built-in XSLT 1.0 and XSLT 2.0 processors are compliant with the relevant W3C drafts. They enable you to transform XML documents directly from within the IDE using either XSLT 1.0 or XSLT 2.0 stylesheets, and to debug XSLT stylesheets using the XSLT Debugger.

Built-in XQuery 1.0 processor

The built-in XQuery 1.0 processor is compliant with the XQuery 1.0 W3C Working Draft of 23 July 2004. It enables you to execute and debug XQuery documents directly from within the IDE.

Transformations of XML documents

XML documents can be transformed directly from within the IDE, either with the internal (built-in) XSLT processor or with any external XSLT processor. To generate PDF from within the IDE, you can specify an external FO processor, and also transform XML to PDF with a single-click after specifying the XSLT stylesheet to use. Furthermore, parameter values can be passed to the XSLT transformation from the IDE itself.

XPath Evaluation

For a given XML document, the Evaluate XPath feature lists the sequence (or nodeset) returned by an XPath expression. You can use either the Document Node or the selected element as the context node. The Evaluate XPath feature is useful if you are creating an XSLT stylesheet and need to evaluate an XPath expression for use in it. You can also navigate to each node in the sequence using the list of nodes in the returned sequence.

XSLT Debugger for XSLT 1.0 and XSLT 2.0

The XSLT Debugger for XSLT 1.0 and XSLT 2.0 is compliant with the relevant W3C drafts. You can use the XSLT Debugger to debug an XSLT stylesheet. The Debugger runs the XSLT stylesheet to be debugged on an XML file. The output is generated step-by-step for each step of the transformation, and you are able to see the context node, template being executed, and other details of each step in the transformation.

XQuery 1.0 Debugger

The XQuery 1.0 Debugger is compliant with the XQuery 1.0 W3C Working Draft of 23 July 2004. It is similar to the XSLT Debugger, and can be used to debug XQuery documents.

XML project management

The XMLSpy IDE enables you to organize related files into projects that are displayed as trees. A project can consist of schema files, XML data files, transformation files, and output files. A project's files are listed together in the Project window, thus enabling easy access to all the project's files when you are working on a project. Furthermore, you can set project-wide or folder-wide specifications, such as the schema file to validate with or the transformation file to use.

Authentic View

Authentic View is a graphical view of an XML document in XMLSpy. Users can enter data into the XML document as they would in a word processor. The StyleVision Power Stylesheet that is used to specify the formatting of the XML document in Authentic View and how the data is input is created in Altova's StyleVision product. Note that Authentic View is also available in Altova's Authentic Desktop, which is currently available free of charge.

Database import

You can import database data as an XML file and generate an XML Schema file to correspond to the database structure. Import of the following databases is supported: MS Access, MS SQL Server, Oracle, MySQL, Sybase, IBM DB2.

Comparing XML files (differencing)

The Compare feature in XMLSpy enables you to detect differences between two XML files. You can set a variety of options to configure the comparison, such as ignoring the order of attributes or child elements, whether entities are resolved or namespaces ignored. The Compare feature can also be used to compare folders.

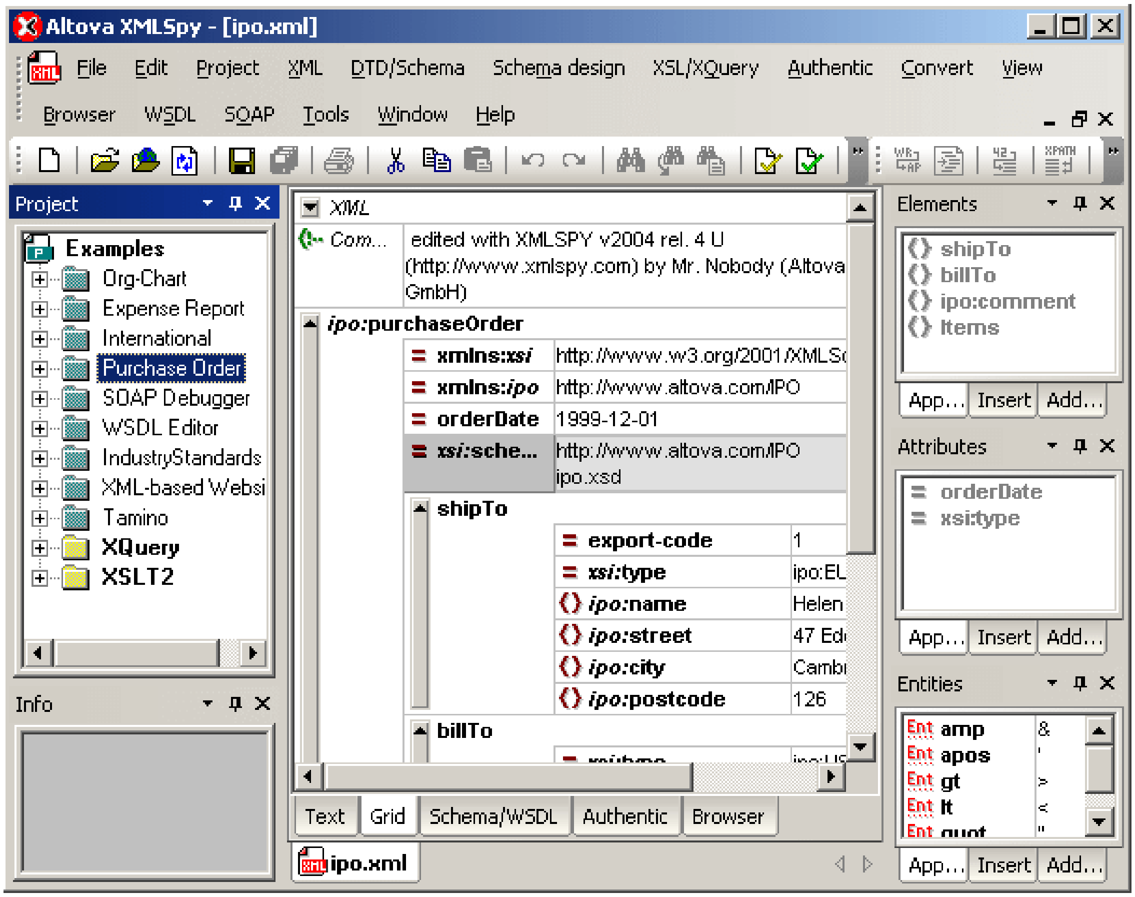
Visual Studio .NET integration

XMLSpy can be integrated in your Visual Studio .NET environment if you would like. To do this, all you need to do is download an executable from the Altova website and run it.

1.3 User interface

XMLSpy has a graphical user interface (*see screenshot*), which is organized into three broad parts:

- **Project Window**, which allows you to organize and edit files and groups of files into projects; and **Info Window**, which displays meta information about the document item being currently edited.
- **Main Window**, where the document you edit appears. The number of document views available in the Main Window depends on the type of document being edited. You can switch between views whenever you want.
- **Entry Helper Windows**, which vary according to the type of document being viewed and the view selected in the Main Window. The Entry Helpers help you to graphically edit your document.



edited with XMLSPY v2004 rel. 4 U (http://www.xmlspy.com) by Mr. Nobody (Altova GmbH)

xmlns:xsi	http://www.w3.org/2001/XMLSc
xmlns:ipo	http://www.altova.com/IPO
orderDate	1999-12-01
xsi:sche...	http://www.altova.com/IPO ipo.xsd

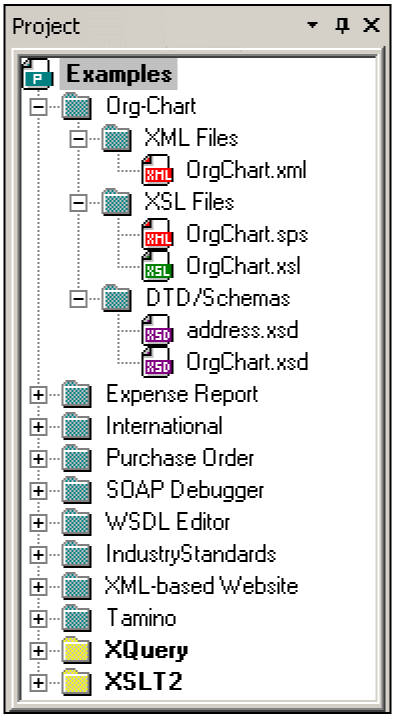
export-code	1
xsi:type	ipo:EU
ipo:name	Helen
ipo:street	47 Edi
ipo:city	Cambi
ipo:postcode	126

These windows can be docked under a menu bar and toolbar (*see screenshot*), or they can be freely arranged under the menu bar and toolbar. Their positions and sizes can be changed by dragging and re-sizing them, as well as by using the toggle on/off commands in the **Window** menu.

This section provides an introduction to these broad parts of the interface. Detailed descriptions of the various interface parts follow this section.

1.3.1 Project Window

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 to allow easy structuring and batch manipulation.



Folders can correspond to physical directories on your file system, or you can define file-type extensions for each folder to keep common files in one convenient place. 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.

Assigning XSL transformations to project folders
You can assign different XSL transformation parameters to each folder and even have the same physical file present in more than one project folder. This is especially useful when you want to keep your data in one XML file and use different XSL stylesheets to produce different output (e.g. separate HTML and WML presentations).

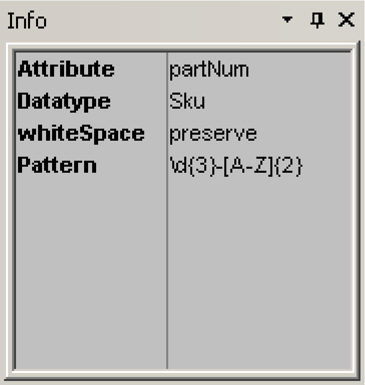
Assigning DTDs / Schemas to project folders
You can assign different DTDs or Schemas to different folders. This allows you to validate a file against both a DTD and an XML Schema without changing the file itself, which is useful when you are in the process of making the transition from DTDs to Schemas.

To manage projects, use the commands in the Project menu.

Please note: You can turn the display of the Project Window on or off with the menu option **Window | Project window**.

1.3.2 Info Window

XMLSpy provides a handy information window that shows detailed information about the element or attribute in which the cursor is currently positioned.



Attribute	partNum
Datatype	Sku
whiteSpace	preserve
Pattern	\d{3}-[A-Z]{2}

This information is available in all editing views and is especially helpful when used in conjunction with the `xsd:annotation` feature.

Please note: You can turn the display of the Info Window on or off with the menu option **Window | Info window**.

1.3.3 Main Window

The Main Window is where you view and edit all documents in XMLSpy.

Managing multiple open files

- You can open and edit any number of XML documents at one time in XMLSpy.
- Each open file in the Main Window is opened in its own Document Window, and each has a tab with its name in it at the bottom of the Main Window.
- When open files are cascaded, tiled, or minimized, a title bar with (i) the name of the file, and (ii) standard minimize, maximize, and close buttons is displayed.
- Open files can be maximized or minimized by clicking the **Maximize** or **Minimize** button, respectively, in the title bar of any open file.
- When you maximize one file, all open files are maximized.
- Open files can be cascaded or tiled using commands in the Window menu.
- To make a file active (in order to edit it), click the file's tab or any part of its window. Alternatively, in the **Window** menu, select the required file from the list of currently open files at the bottom of the menu.
- You can also activate open files in the sequence in which they were opened by using **Ctrl+Tab** or **Ctrl+F6**.

Accessing File commands quickly

To access File commands quickly (such as printing, closing, sending as an e-mail attachment), right-click the file's tab. This opens a context-menu with a selection of File commands.

Main Window Views

XMLSpy provides multiple views of your XML document. These views are either editing or browser views:

- **Text View:** An editing view with syntax-coloring for source-level work
- **Enhanced Grid View:** For structured editing. The document is displayed as a structured grid, which can be manipulated graphically. This view also contains an embedded **Database/Table view**, which shows repeating elements in a tabular format
- **Schema/WSDL View:** For viewing and editing XML Schemas

- **Authentic View:** For editing XML documents based on StyleVision Power Stylesheets
- **Browser View:** An integrated browser view that supports both CSS and XSL stylesheets.

To switch between document views in the Document Window, click on the appropriate view button at the bottom of the Document Window. Alternatively, use the commands in the View menu.

Please note: You can customize the default view (that is, the Main Window view) for individual file extensions. To do this, go to the Tools | Options dialog, and make the required settings in the **File types** and **View** tabs.

1.3.4 Entry Helpers

XMLSpy has intelligent editing features that help you to create valid XML documents quickly. These features are organized into three palette-like windows we call **Entry Helpers**.

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, as well as information about these.

The Entry Helper windows have an XMLSpy prefix in Visual Studio .NET.

Entry Helpers in different views
What Entry Helpers are displayed depend upon the view. The different sets of Entry Helpers are categorized as follows, according to the views available in your Altova product:

- Text View and Grid View: Elements, Attributes, and Entities Entry Helpers
- Schema Design View: Component Navigator, and Details and Facets Entry Helpers
- Authentic View: Elements, Attributes, and Entities Entry Helpers

The Entry Helpers for each view are described in the section about that view (in the following sections of this documentation).

Please note: You can turn the display of Entry Helpers on or off with the menu option **Window | Entry Helpers**.

1.3.5 Menu Bar and Toolbar

Menu Bar
The menu bar 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 shorcut key **F8** to validate an XML file.

Toolbar

The toolbar contains buttons that are shortcuts for commands found in the menus. The name of the command appears when you place your mouse pointer over the button. To execute the command, click the button.

Toolbar buttons are arranged in groups. In the **Tools | Customize | Toolbars** dialog, you can specify which toolbar groups are to be displayed. In the GUI, you can also drag toolbar groups to alternative locations by clicking and dragging a toolbar handle to the desired location.

2 Tutorials

This section contains the following tutorials:

- XMLSPY Tutorial: A tutorial that runs you through the creation of an XML Schema; the creation, editing, and transformation of an XML file; the use of databases for XML in XMLSpy; and the organization of related files into XMLSPY Projects.
- Authentic View Tutorial: A tutorial to introduce you to Altova's unique Authentic View interface and its features. In Authentic View, you can edit XML documents in a graphical interface without having to know XML or the structure of the schema on which the document is based.

2.1 XMLSpy Tutorial

This tutorial gives a short overview of XML and takes you through several tasks which provide an overview of how to use XMLSpy to its fullest.

You will learn how to:

- Create a **simple schema** from scratch
- **Generalize** the schema using simple and complex types
- Create schema **documentation**
- Create an **XML document** based on the schema file
- Copy XML data to a **third party product** (Excel) and reinsert it in XMLSpy
- **Validate** the XML document against its schema
- **Update** Schema settings while editing the XML document
- **Transform** the XML document into HTML using XSLT, and see the result in the Browser view
- **Import** and **export** database data to and from XMLSpy
- Create a **schema** from an MS Access database
- Create an XMLSpy **project** to organize all 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 `...\Examples\Tutorial` folder.

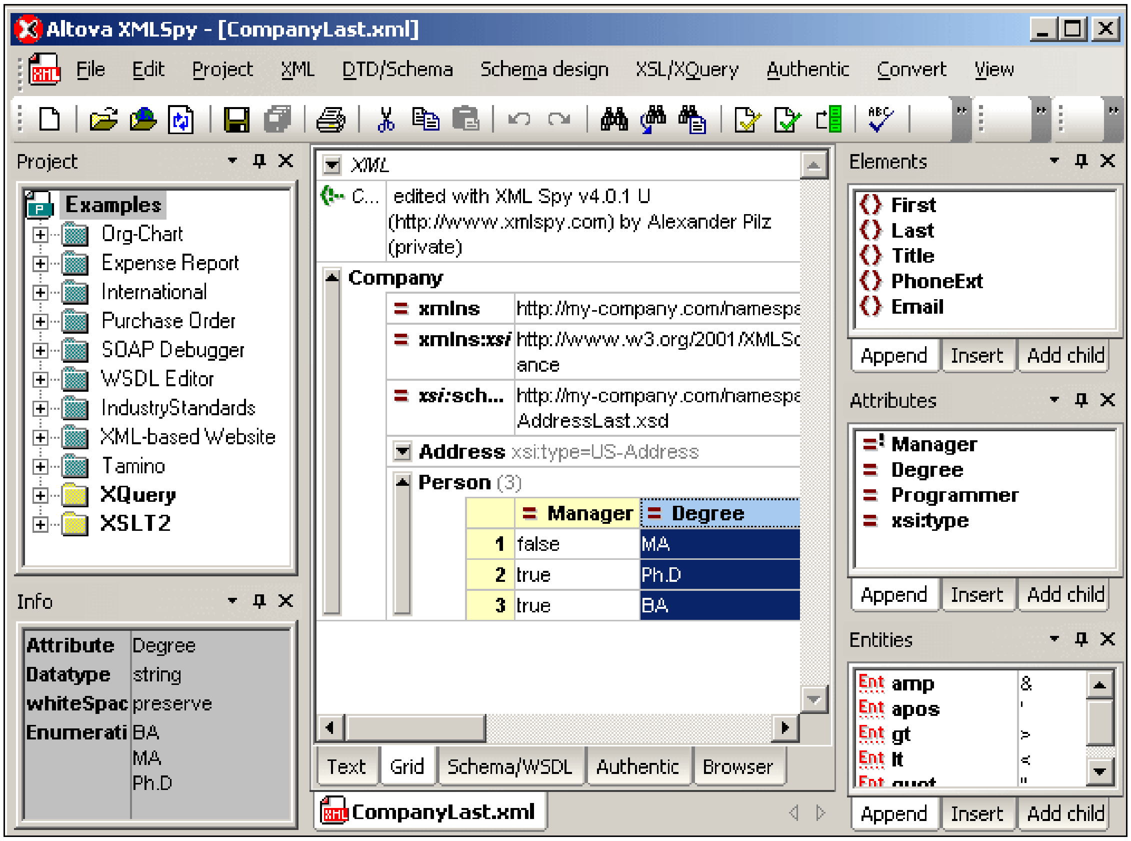
The `Examples` folder contains various XML files for you to experiment with, while the `Tutorial` folder contains all the files used in this tutorial.

The `Template` folder 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.

2.1.1 The 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/WSDL Design View, Authentic View, and Browser View. These views are described in detail in the respective sections on them.
- The right 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.



C... edited with XML Spy v4.0.1 U (http://www.xmlspy.com) by Alexander Pilz (private)

Company
= xmlns http://my-company.com/namespa
= xmlns:xsi http://www.w3.org/2001/XMLSchance
= xsi:sch... http://my-company.com/namespa AddressLast.xsd
Address xsi:type=US-Address
Person (3)

	= Manager	= Degree
1	false	MA
2	true	Ph.D
3	true	BA

Attribute	Degree
Datatype	string
whiteSpac	preserve
Enumerati	BA MA Ph.D

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.

2.1.2 Creating a basic XML Schema

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, the Schema/WSDL Design 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/WSDL View.

Objective
In this section of the tutorial, you will learn how to edit XML Schemas in Schema/WSDL 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 the Schema/WSDL View exclusively. The following commands are used:

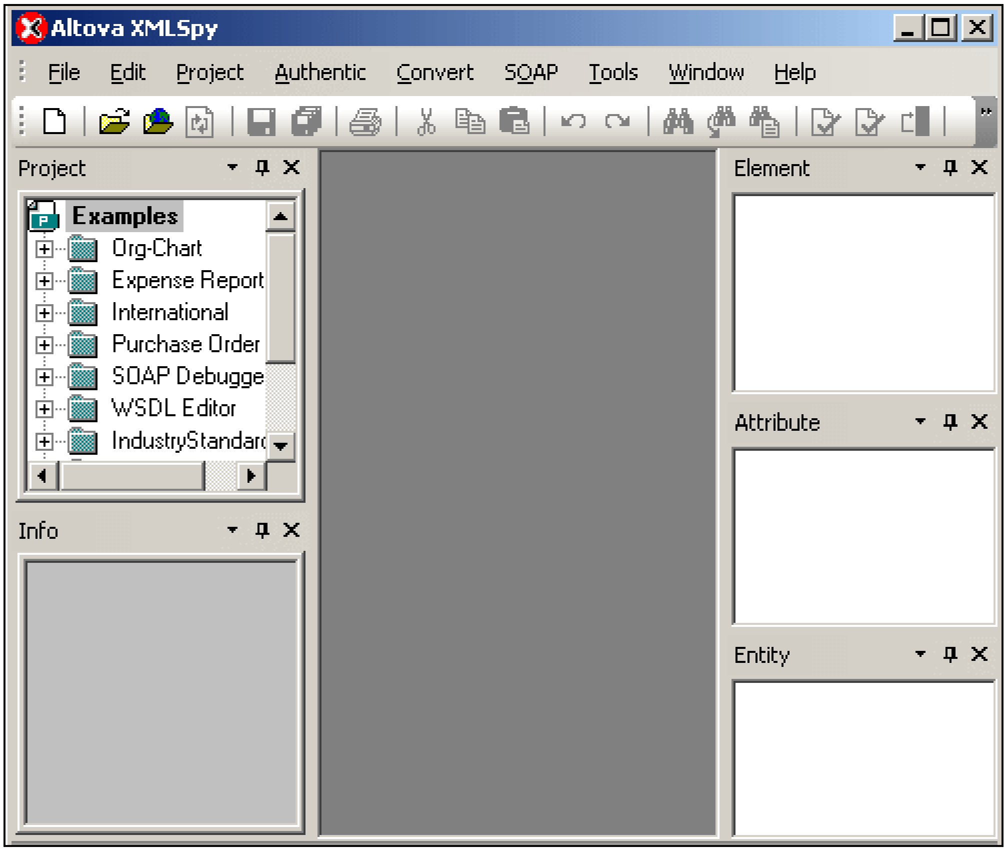
Display Diagram (or Display Content Model View). This icon is located to the left of all global components in Schema Overview. Click this icon to display the content model of the associated global component.

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.

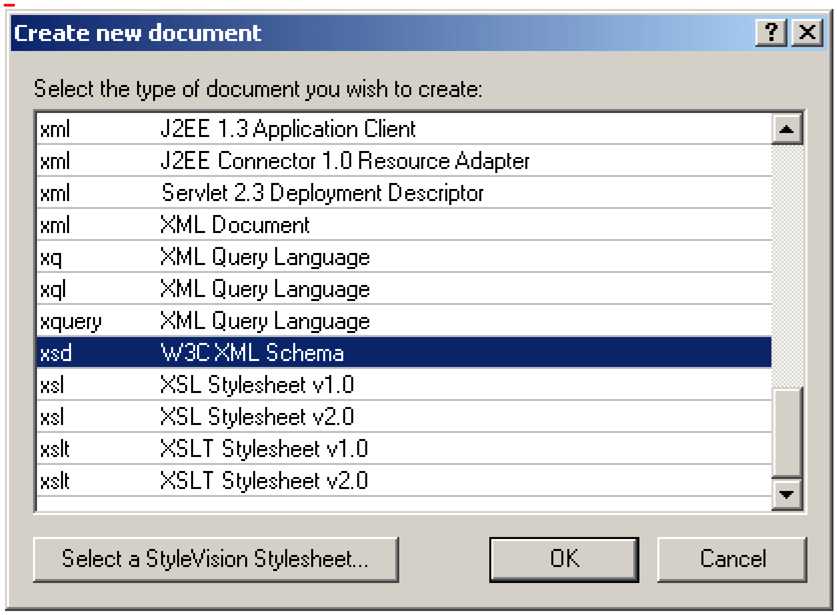


Note the three main parts of the interface: (i) the Project and Info Windows on the left; (ii) the Main Window in the middle; and (iii) the Entry Helpers on the right.

Creating a new XML Schema file

To create a new XML Schema file:

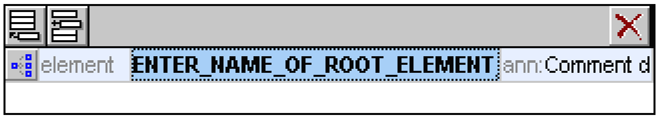
1. Select the menu option **File | New**. The Create new document dialog opens.



Select the type of document you wish to create:

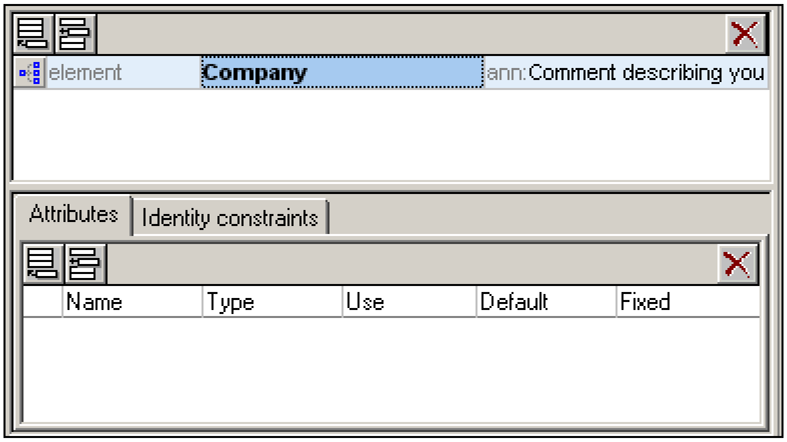
xml	J2EE 1.3 Application Client
xml	J2EE Connector 1.0 Resource Adapter
xml	Servlet 2.3 Deployment Descriptor
xml	XML Document
xq	XML Query Language
xql	XML Query Language
xquery	XML Query Language
xsd	W3C XML Schema
xsl	XSL Stylesheet v1.0
xsl	XSL Stylesheet v2.0
xslt	XSLT Stylesheet v1.0
xslt	XSLT Stylesheet v2.0

2. In the dialog, select the `xsd W3C XML Schema` entry, and confirm with **OK**. An empty schema file appears in the Main Window in Schema/WSDL Design View. You are prompted to enter the name of the root element.



element ENTER_NAME_OF_ROOT_ELEMENT ann:Comment d

3. 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).

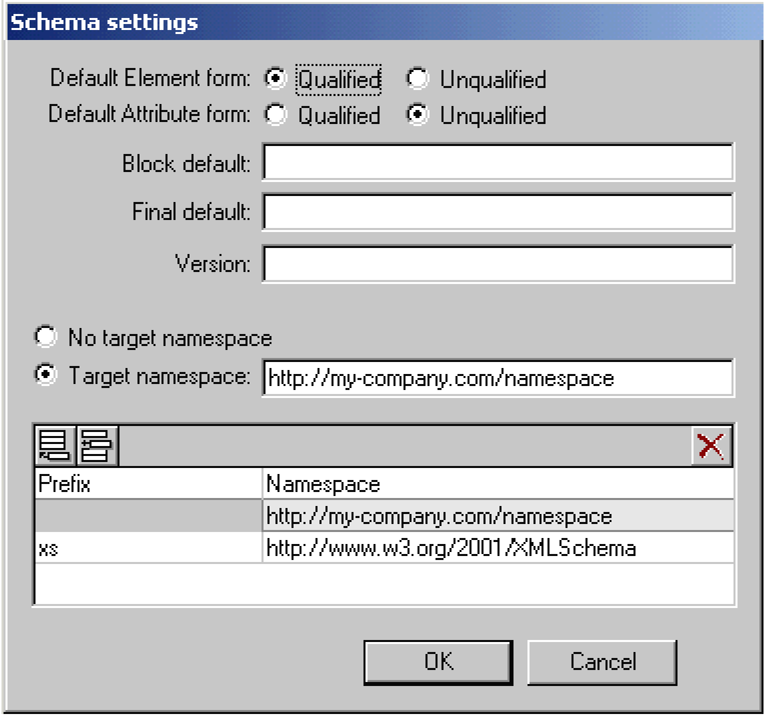
Defining namespaces

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.



Schema settings

Default Element form: ● Qualified ○ Unqualified

Default Attribute form: ○ Qualified ● Unqualified

Block default:

Final default:

Version:

○ No target namespace

● Target namespace: http://my-company.com/namespace

Prefix	Namespace
	http://my-company.com/namespace
xs	http://www.w3.org/2001/XMLSchema

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.

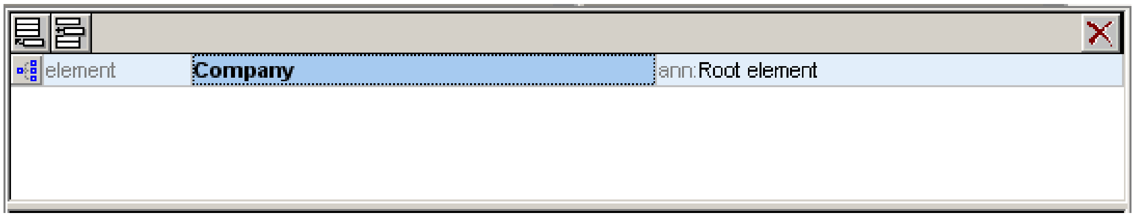
Please note:

- 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.

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 the Schema/WSDL 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.



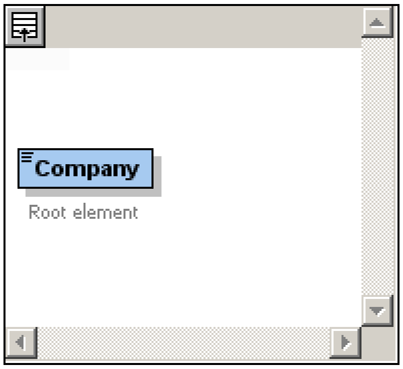
element	**Company**	ann: Root element

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

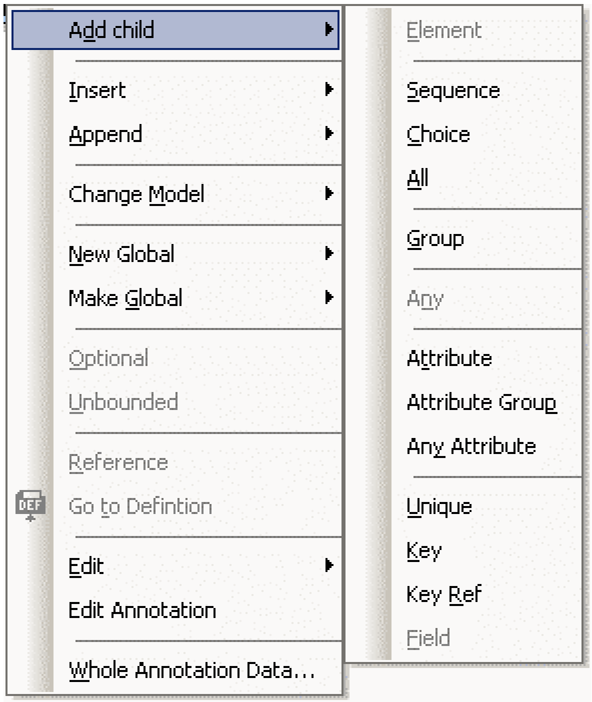
Creating a basic content model

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

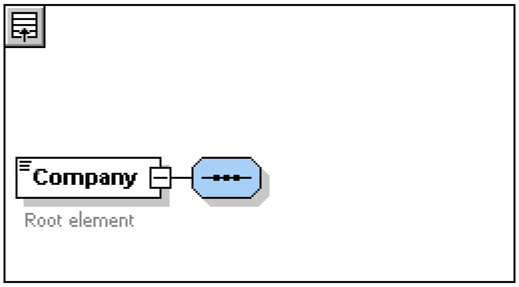
1. In the Schema Overview, click the Display Diagram icon of the `Company` element. This displays the content model of the `Company` element, which is currently empty. Alternatively, you can double-click the `Company` entry in the Components entry helper to display its content model.



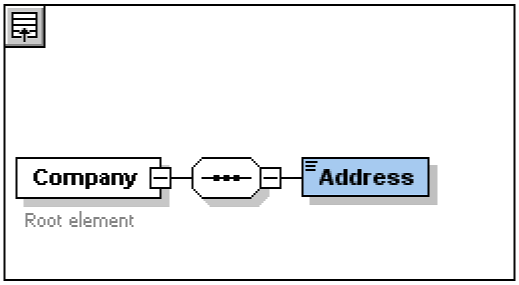
2. A content model consists of **compositors** and **components**. The compositors specify the relationship between two components. At this point of the `Company` content model, you must add a child compositor to the `Company` element in order to add a child element. To add a compositor, right-click 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.)



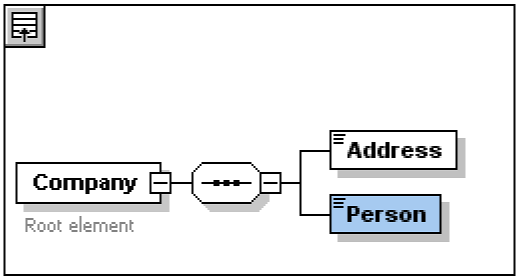
This inserts the Sequence compositor, which defines that the components that follow must appear in the specified sequence.



3. Right-click the Sequence compositor and select **Add Child | Element**. An unnamed element component is added.
4. Enter `Address` as the name of the element, and confirm with **Enter**.

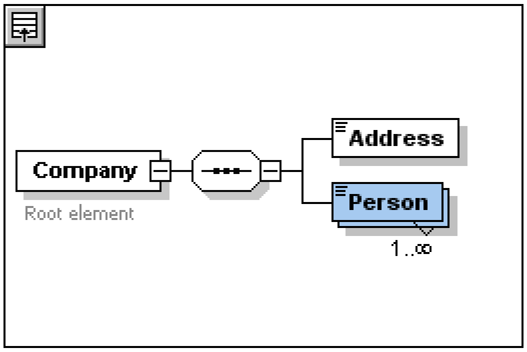


5. Right-click the Sequence compositor again, select **Add Child | Element**. Name the newly created element component `Person`.



You have so far defined a schema which allows for one address and one person per company. We need to increase the number of `Person` elements.

6. Right-click the `Person` element, and select **Unbounded** from the context menu. The `Person` element in the diagram now shows the number of allowed occurrences: 1 to infinity.



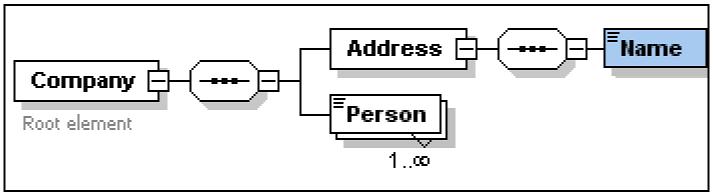
Alternatively, in the Details Entry Helper, you can edit the `minOcc` and `maxOcc` fields to specify the allowed number of occurrences, in this case 1 and unbounded, respectively.

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.

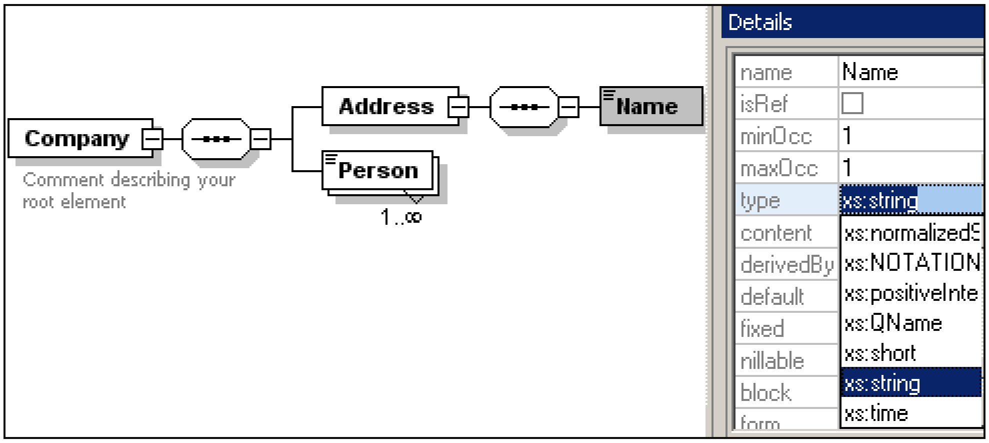
```
<xs:element name="Company">
  <xs:annotation>
    <xs:documentation>Root element</xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Address">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Name"/>
          </xs:sequence>
        </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/WSDL** tab to return to Schema/WSDL 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"/>
```

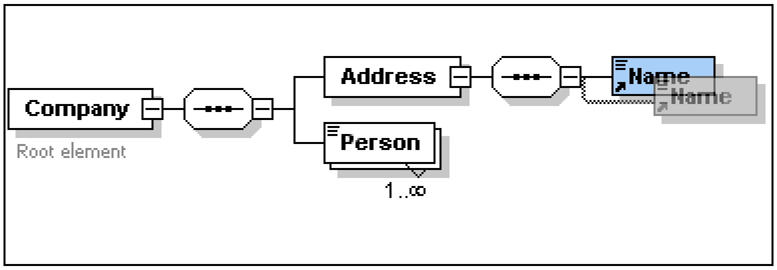
Please note: A simple type element can have any one of several XML Schema data types. In all these cases, the icon indicating text-content appears in the element box.

Adding elements with drag-and-drop

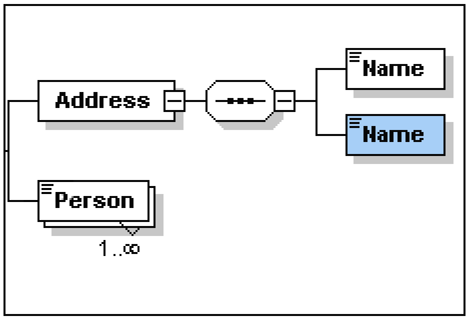
You have added elements using the context menu that appears when you right-click an element or compositor. You can also create elements using drag-and-drop, which is quicker than using menu commands. In this section, you will add more elements to the definition of the `Address` element using drag-and-drop,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

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

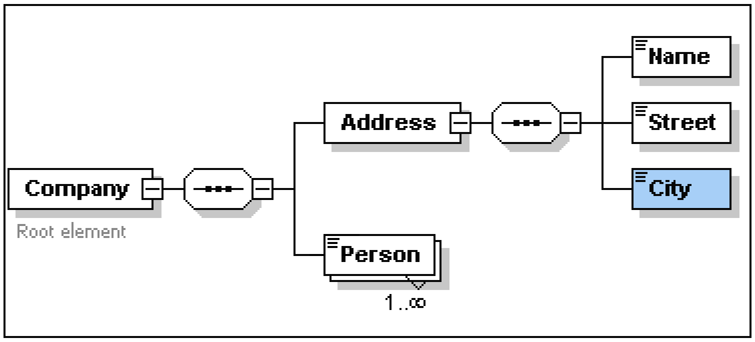
1. Click the `Name` element of the `Address` element, hold down the **Ctrl** key, and drag the element box with the mouse. A small "plus" icon appears in the element box, indicating that you are about to copy the element. A copy of the element together with a connector line also appears, showing where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



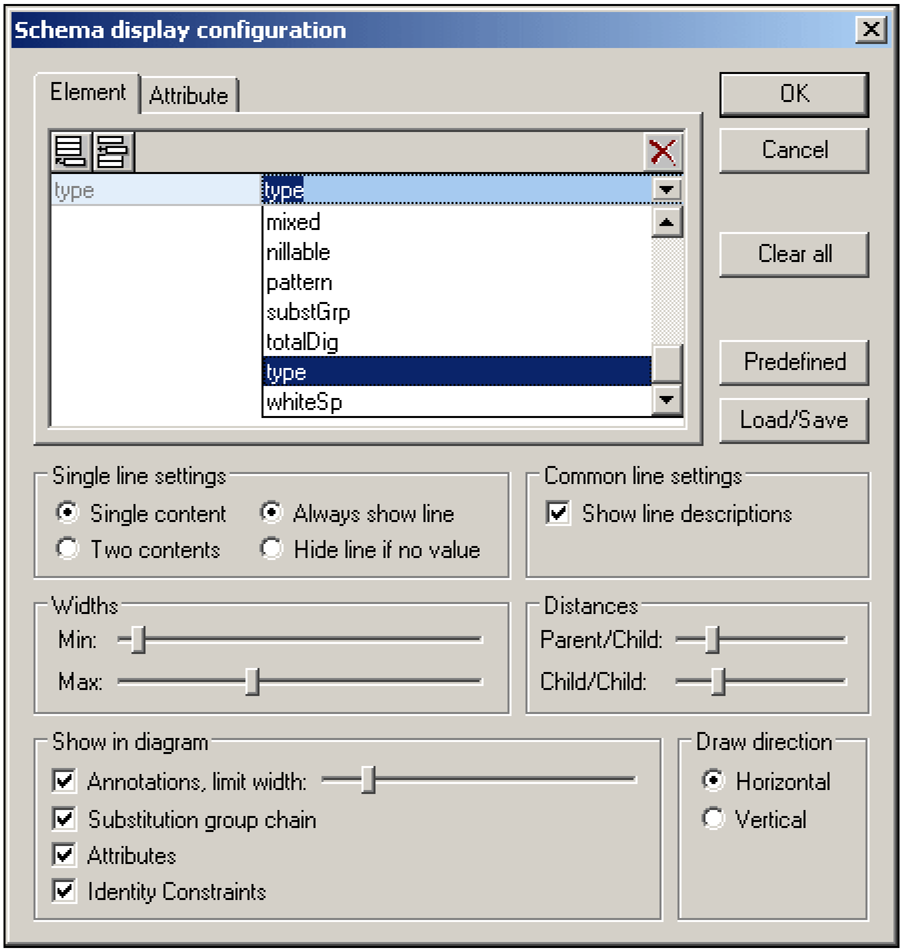
The `Address` element has a sequence of a `Name`, a `Street`, and a `City` element, in that order.

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.



Schema display configuration

Element | Attribute

type	type

mixed
nillable
pattern
substGrp
totalDig
type
whiteSp

OK

Cancel

Clear all

Predefined

Load/Save

Single line settings

● Single content ● Always show line

○ Two contents ○ Hide line if no value

Common line settings

☑ Show line descriptions

Widths

Min:

Max:

Distances

Parent/Child:

Child/Child:

Show in diagram

☑ Annotations, limit width:

☑ Substitution group chain

☑ Attributes

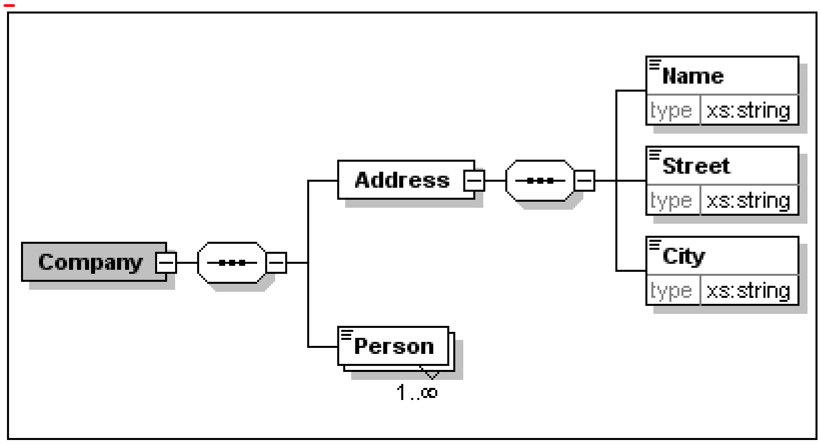
☑ Identity Constraints

Draw direction

● Horizontal

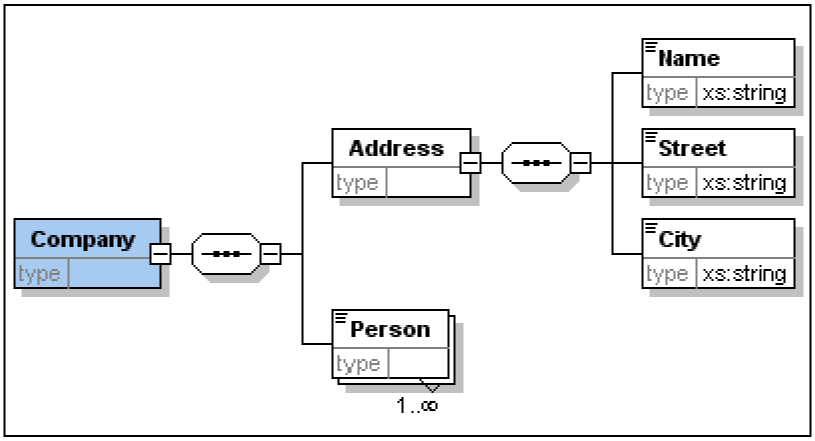
○ Vertical

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:

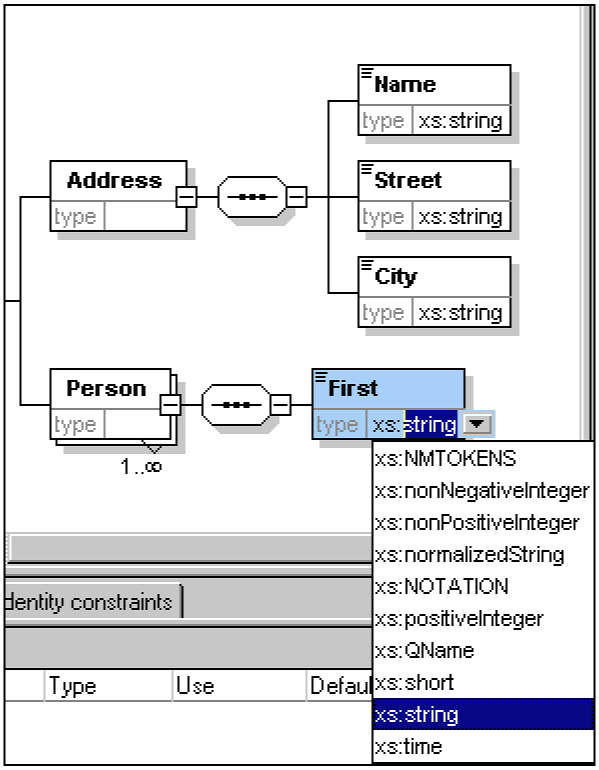
- The property descriptor lines are editable, so values you enter in them become part of the element definition.
- The settings you define in the Schema display configuration dialog apply to the schema documentation output as well as the printer output.

Completing the basic schema

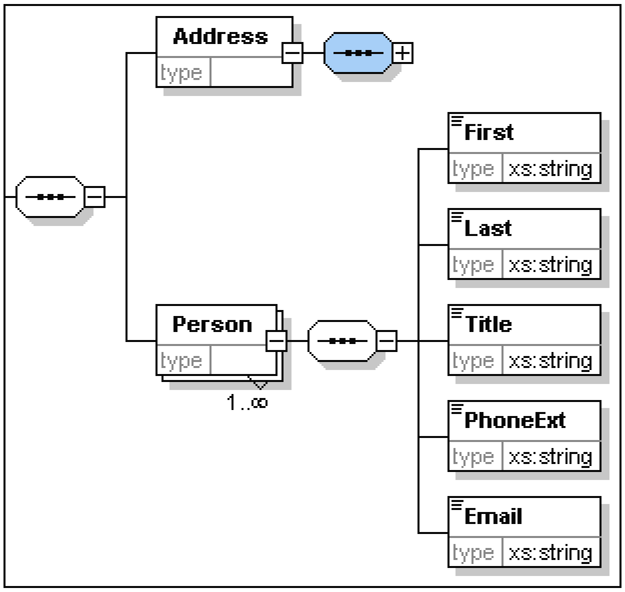
You have defined the content of the `Address` element. Now you need to define the content of the `Person` element. The `Person` element is to contain the following child elements, all of which are simple types: `First`, `Last`, `Title`, `PhoneExt`, and `Email`. All these elements are mandatory except `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.



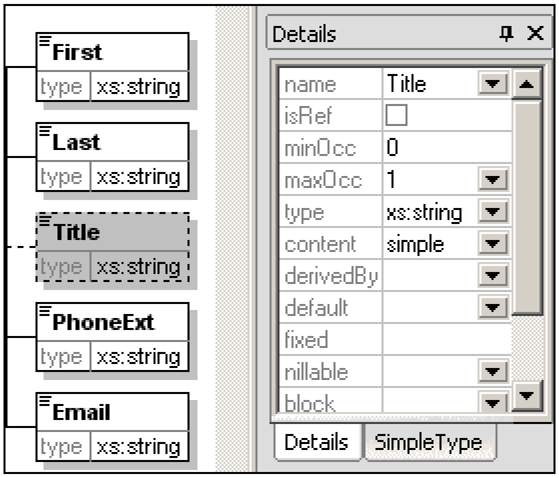
4. Select the `xs:string` entry from the dropdown list or enter it into the `type` value field.
5. Use the drag-and-drop method to create four more elements. Name them `Last`, `Title`, `PhoneExt`, and `Email`, respectively.



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.



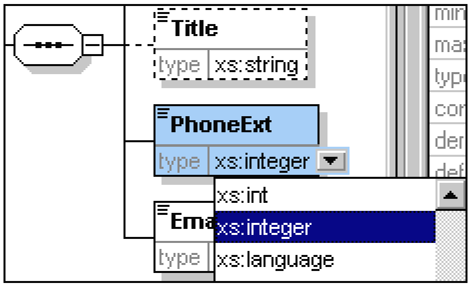
name	Title
isRef	□
minOcc	0
maxOcc	1
type	xs:string
content	simple
derivedBy	
default	
fixed	
nillable	
block	

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

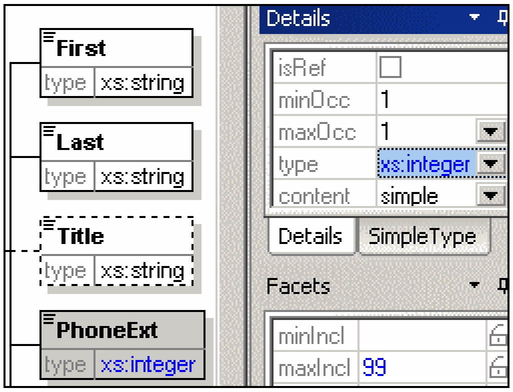
To define the `PhoneExt` element to be of type `xs:integer` and have a maximum of two digits:

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**.



isRef	□
minOcc	1
maxOcc	1
type	xs:integer
content	simple

minIncl	
maxIncl	99

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 `Examples\Tutorial` folder of your XMLSpy application folder.

2.1.3 Advanced XML Schema definitions

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 the Schema/WSDL 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.

Working with Complex Types and Simple Types

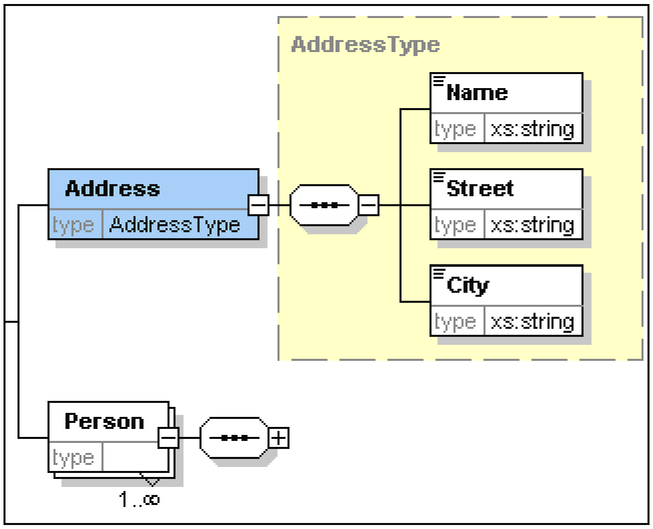
Having defined the content model of an element, you may decide you want to reuse it elsewhere in your schema. The way to do this is by creating that element definition as a global complex type or as a global element. In this section, you will work with global complex types. You will first create a complex 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

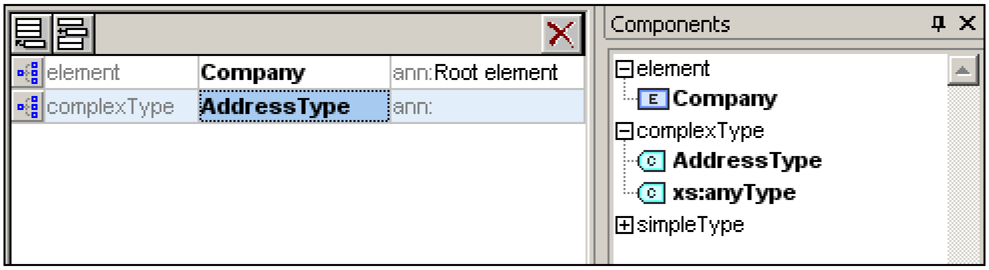
The basic `Address` element that we defined (containing `Name`, `Street`, and `City` elements) can be reused in various address formats. So let us create this element definition as a complex type, which can be reused.

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`.

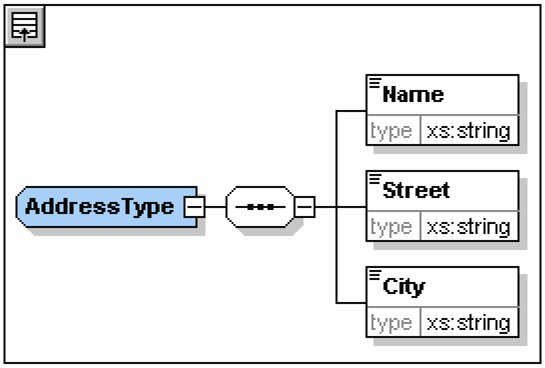


3. Click the Display All Globals icon. This takes you to the Schema Overview, in which you can view all the global components of the schema.
4. Click the expand icons for the **element** and **complexType** entries in the Components entry helper, to see the respective schema constructs.
   The Schema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



element	Company	ann:Root element
complexType	AddressType	ann:

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.

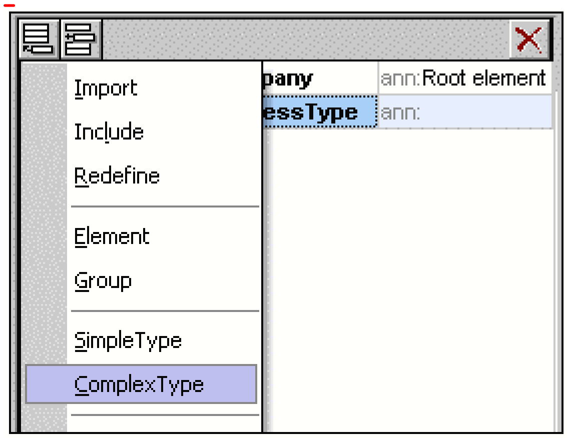


6. Click the Display All Globals icon to return to the Schema Overview.

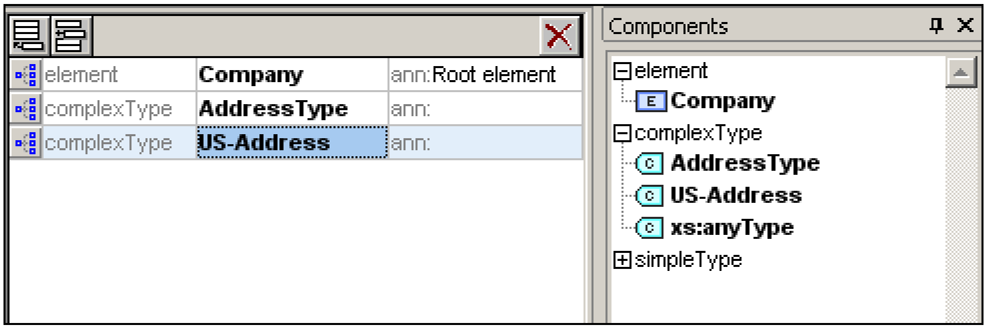
Extending a complex type definition
We now want to use the global `AddressType` component to create two kinds of country-specific addresses. For this purpose we will define a new complex type based on the basic `AddressType` component, and then extend that definition.

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:

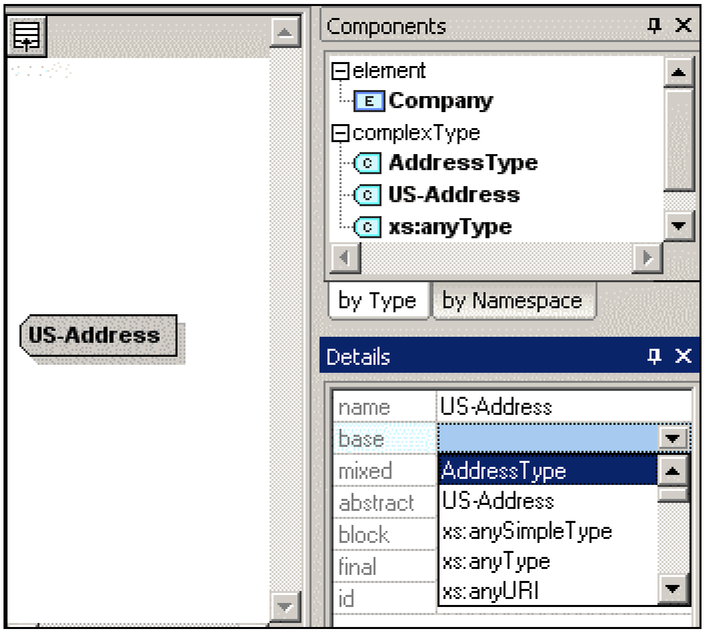


3. Select **ComplexType** from the context menu. A new line appears in the component list, and the cursor is set for you to enter the component name.
4. Enter `US-Address` and confirm with **Enter**. (If you forget to enter the hyphen character "**-**" and enter a space, the element name will appear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



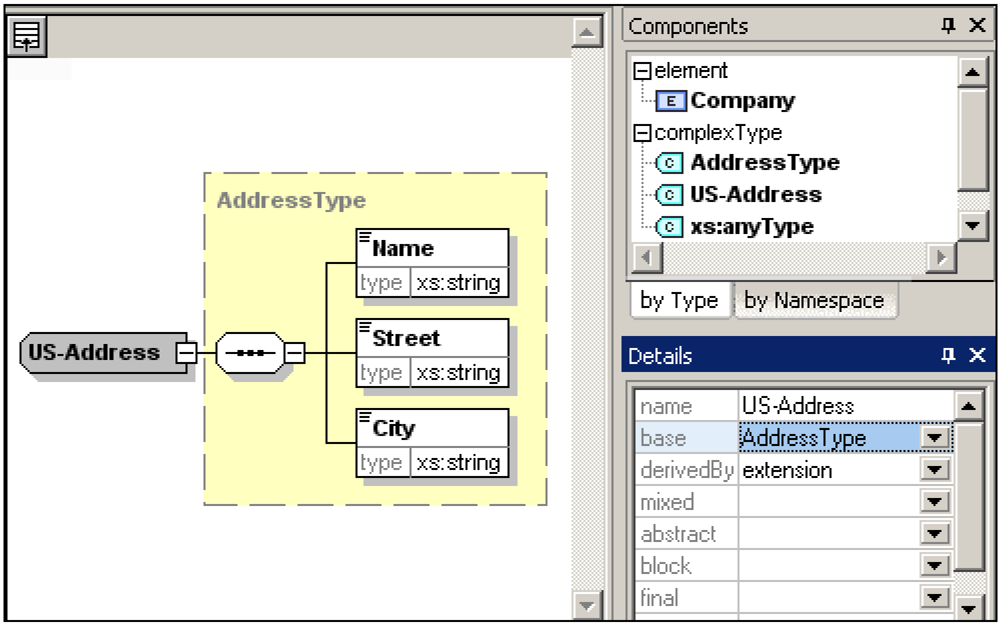
element	**Company**	ann:Root element
complexType	**AddressType**	ann:
complexType	**US-Address**	ann:

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.



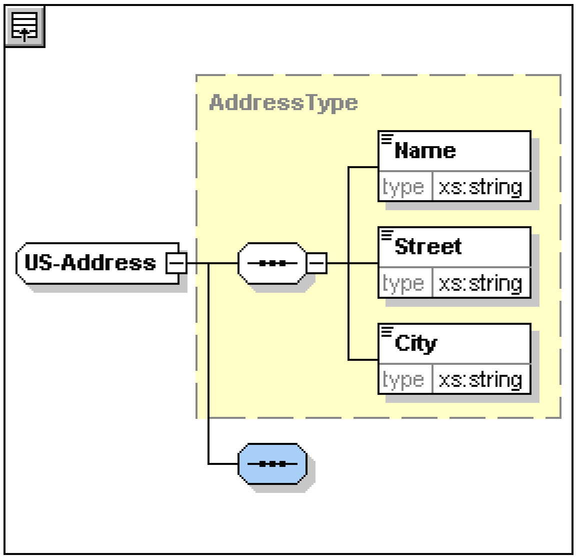
name	US-Address
base	
mixed	AddressType
abstract	US-Address
block	xs:anySimpleType
final	xs:anyType
id	xs:anyURI

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

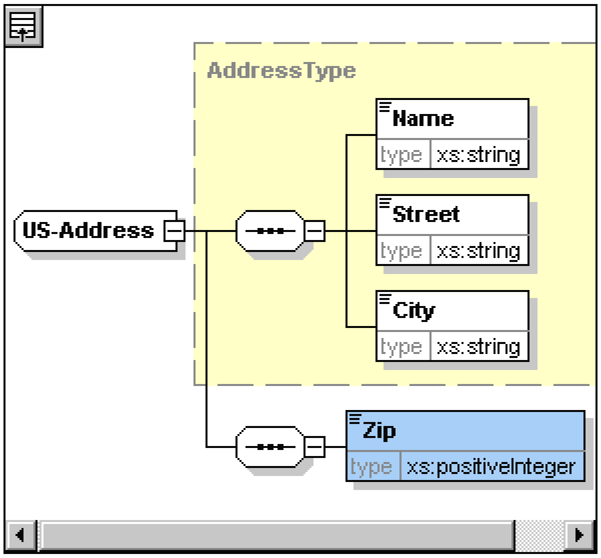


name	US-Address
base	AddressType
derivedBy	extension
mixed	
abstract	
block	
final	

7. 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** button. 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

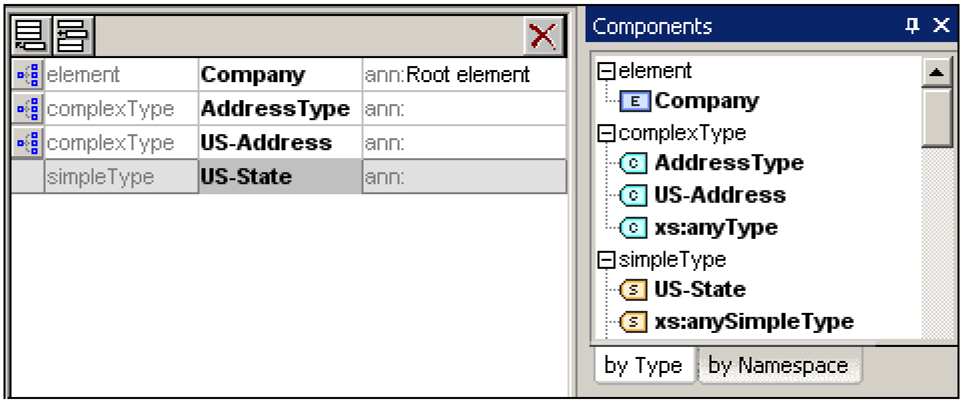
Just as the complex type `US-Address` is based on the complex type `AddressType`, an element can also be based on a simple type. The advantage is the same as for global complex types: the simple type can be reused. In order to reuse a simple type, the simple type must be defined globally. In this tutorial, 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).

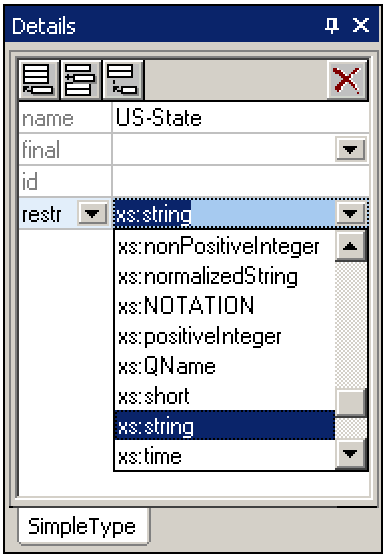


element	**Company**	ann:Root element
complexType	**AddressType**	ann:
complexType	**US-Address**	ann:
simpleType	**US-State**	ann:

Components

- element
  - **Company**
- complexType
  - **AddressType**
  - **US-Address**
  - **xs:anyType**
- simpleType
  - **US-State**
  - **xs:anySimpleType**

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.

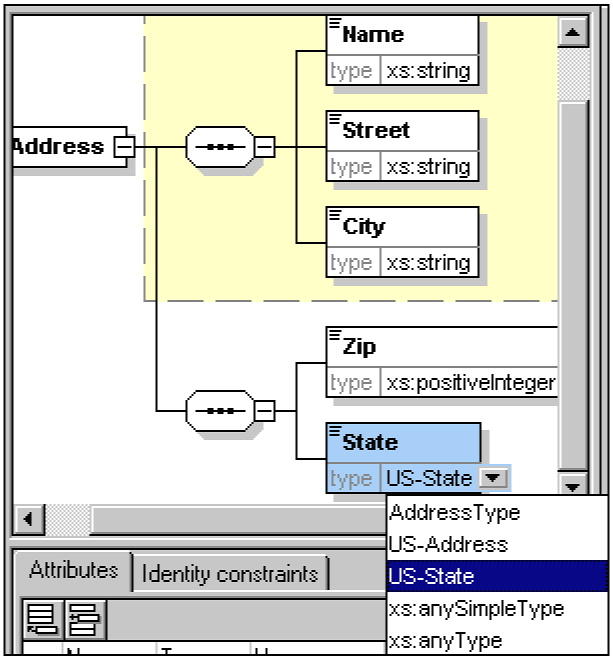


This creates a simple type called `US-State`, which is of datatype `xs:string`. This global component can now be used in the content model of `US-Address`.

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.

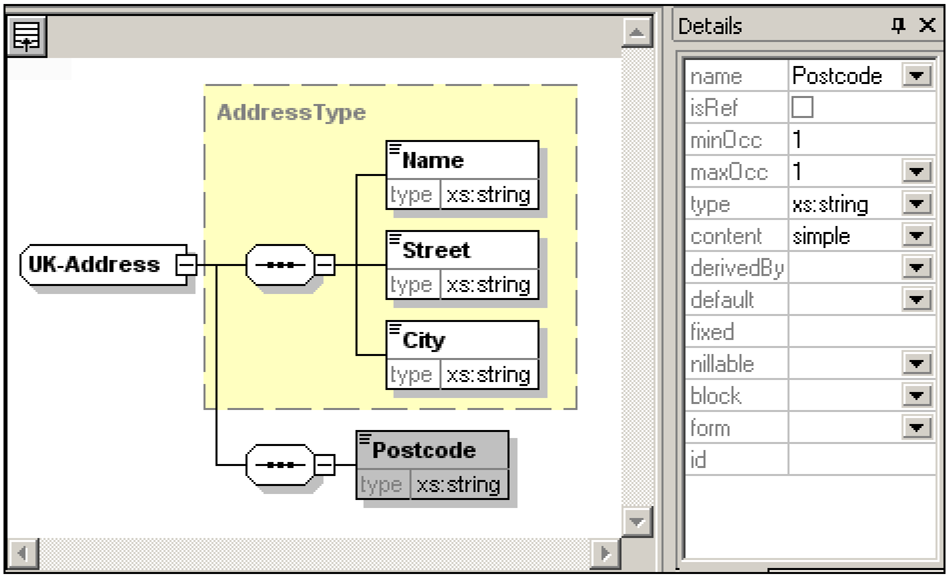
Creating a second complex type based on `AddressType`

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:



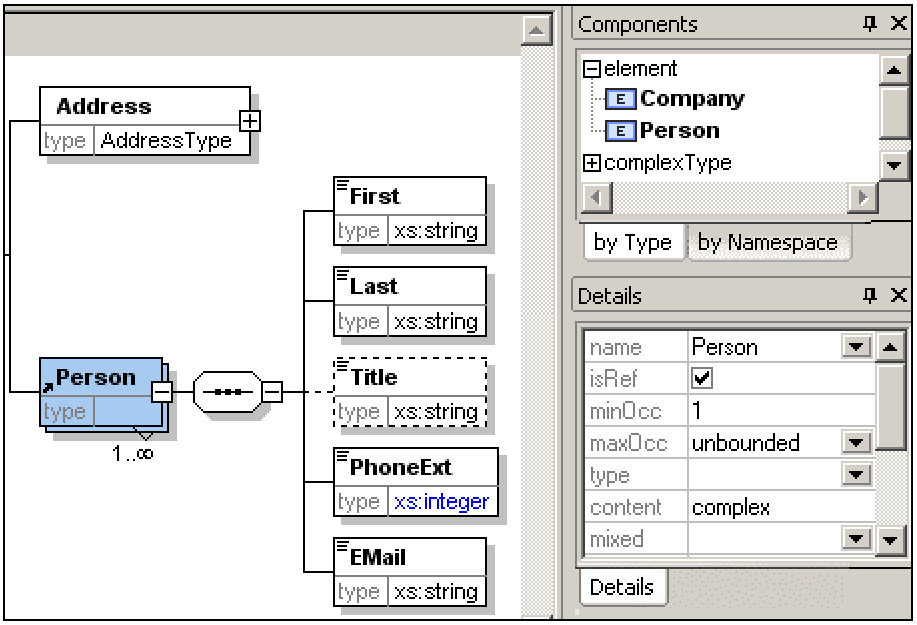
name	Postcode
isRef	□
minOcc	1
maxOcc	1
type	xs:string
content	simple
derivedBy	
default	
fixed	
nillable	
block	
form	
id	

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.

Referencing global elements

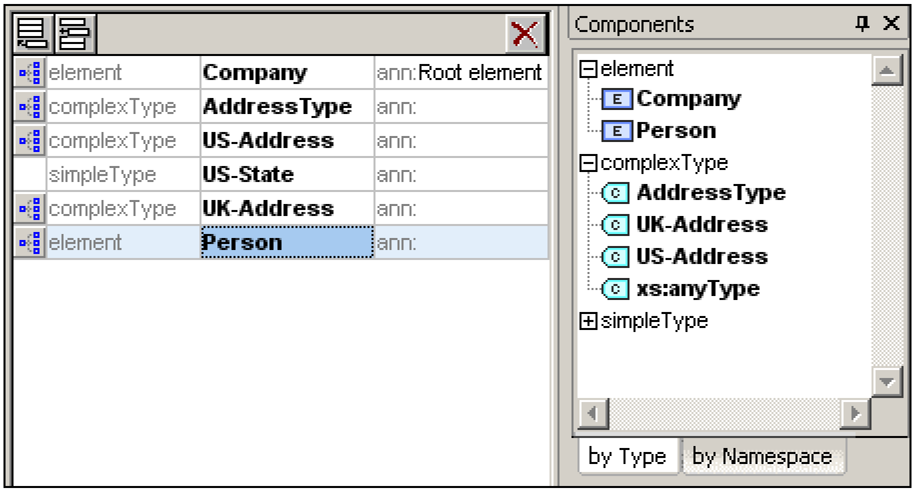
In this section, we will convert the locally defined `Person` element to a global element and reference that global element from within the `Company` element.

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.



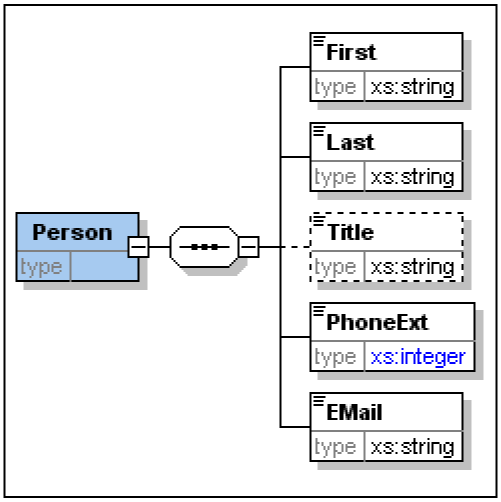
name	Person
isRef	☑
minOcc	1
maxOcc	unbounded
type	
content	complex
mixed	

4. 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.



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

5. In the Components Entry Helper, 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.

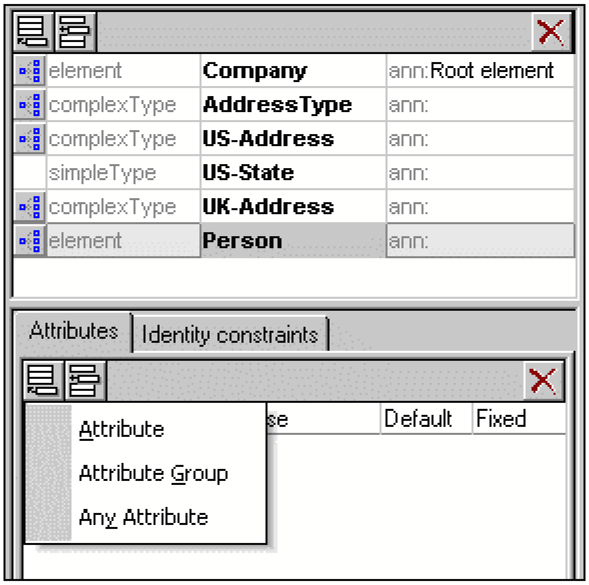
  A globally declared element can be reused at multiple locations. It differs from a globally declared complex type in that its content model cannot be modified without also modifying the global element itself. If you change the content model of an element that references a global element, then the content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

Attributes and attribute enumerations

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

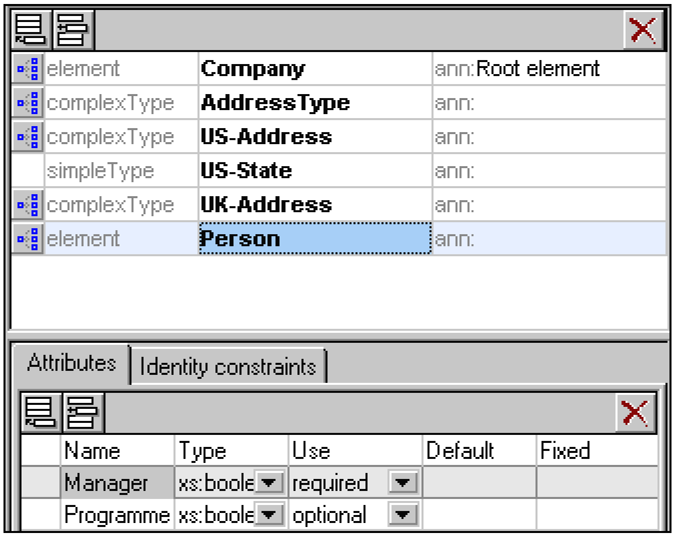
Defining element attributes

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.



element	**Company**	ann:Root element
complexType	**AddressType**	ann:
complexType	**US-Address**	ann:
simpleType	**US-State**	ann:
complexType	**UK-Address**	ann:
element	**Person**	ann:

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`.



element	**Company**	ann:Root element
complexType	**AddressType**	ann:
complexType	**US-Address**	ann:
simpleType	**US-State**	ann:
complexType	**UK-Address**	ann:
element	**Person**	ann:

Name	Type	Use	Default	Fixed
Manager	xs:boole	required		
Programme	xs:boole	optional		

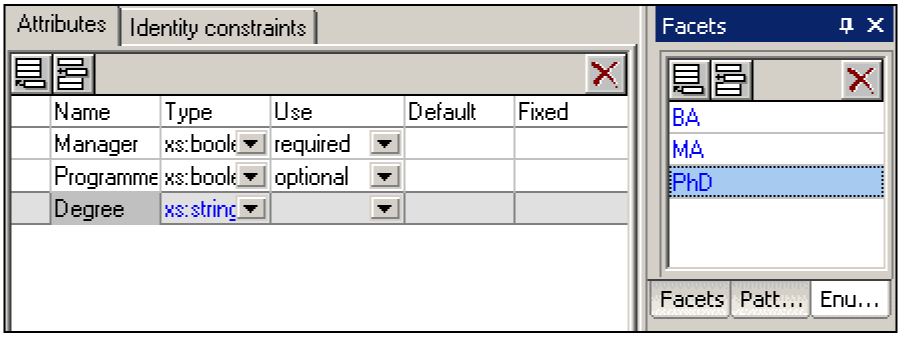
6. Use the same procedure to create a `Programmer` attribute with `Type=xs:boolean` and `Use=optional`.

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:

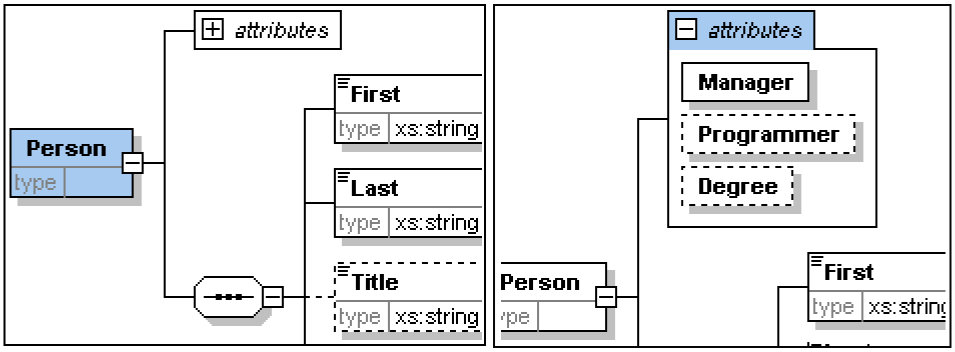
1. In the Schema Overview, click the `Person` element to make it active.
2. Click the Append icon in the top left of the Attributes window, and select the **Attribute** entry.
3. Enter `Degree` as the attribute name, and select `xs:string` as its type.
4. With the Degree attribute selected, in the Facets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



Name	Type	Use	Default	Fixed
Manager	xs:bool	required		
Programme	xs:bool	optional		
Degree	xs:string			

BA
MA
PhD

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 can be toggled by selecting the menu option **Schema Design | Configure view**, and activating the **Attributes** and **Identity Constraints** check boxes 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.

2.1.4 Schema navigation and documentation

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 the Schema/WSDL View exclusively. The following commands are used:

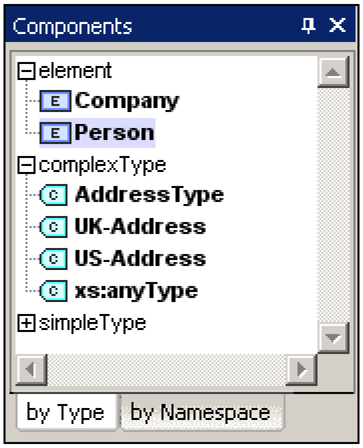
⊞ Display Diagram (or Display Content Model View). This icon is located to the left of all global components in Schema Overview. Clicking the icon causes the content model of the associated global component to be displayed.

Schema navigation

This section shows you how to navigate the Schema/WSDL 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*).



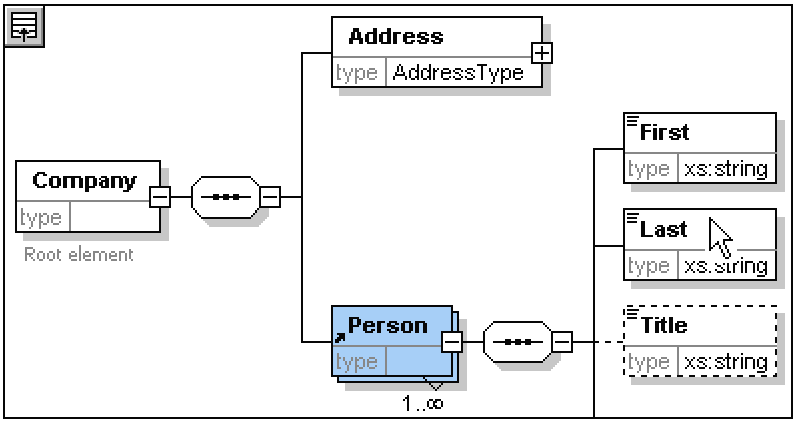
If you double-click any of the other global components (simple type, attribute, attribute group) in the Components Entry Helper, that component will be highlighted in Schema Overview (since they do not have a content model).

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.

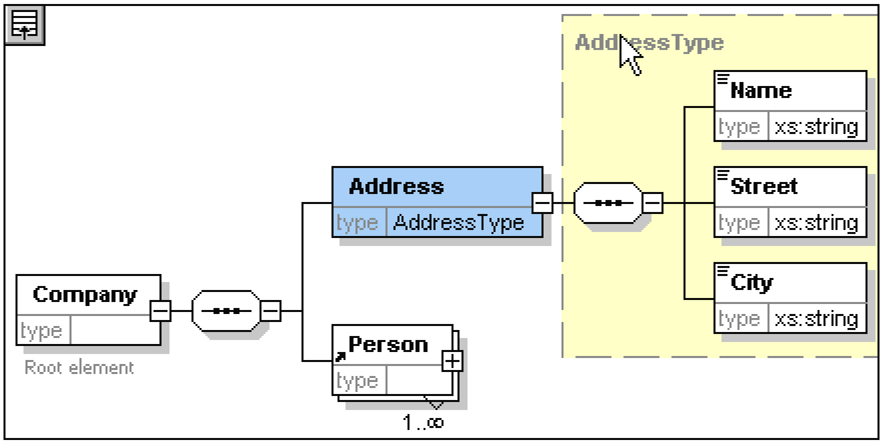
For example, while viewing the `Company` content model, holding down **Ctrl** while double-clicking `Last` opens the `Person` content model and highlights the `Last` element in it.



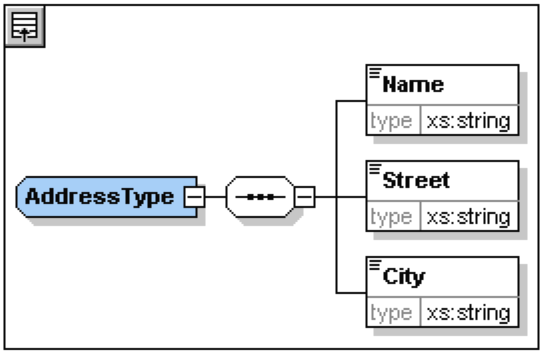
When the `Last` element is highlighted, all its properties are immediately displayed in the relevant entry helpers and information window.

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.

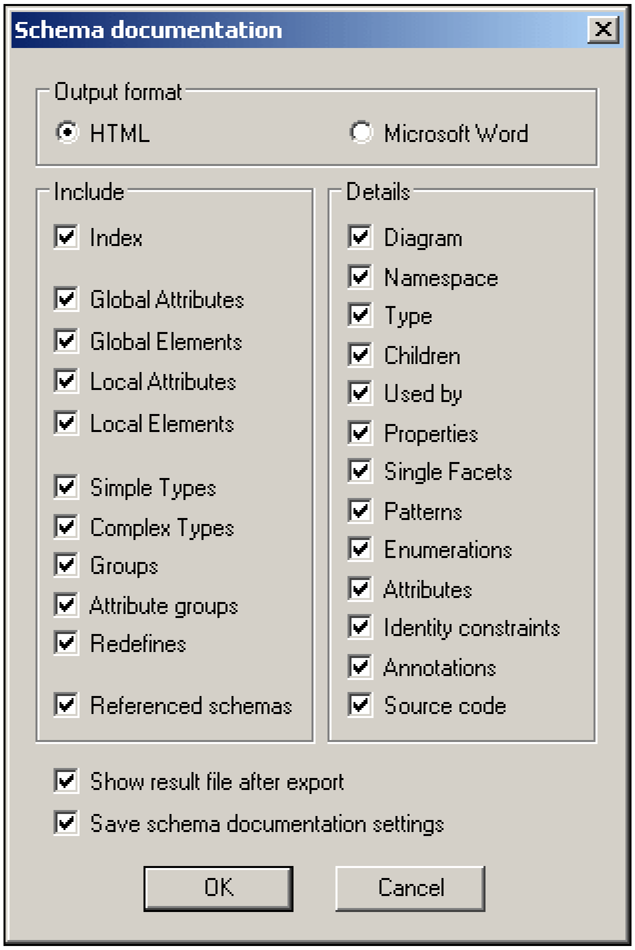
Schema documentation

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.



Output format

● HTML ○ Microsoft Word

Include

☑ Index

☑ Global Attributes

☑ Global Elements

☑ Local Attributes

☑ Local Elements

☑ Simple Types

☑ Complex Types

☑ Groups

☑ Attribute groups

☑ Redefines

☑ Referenced schemas

Details

☑ Diagram

☑ Namespace

☑ Type

☑ Children

☑ Used by

☑ Properties

☑ Single Facets

☑ Patterns

☑ Enumerations

☑ Attributes

☑ Identity constraints

☑ Annotations

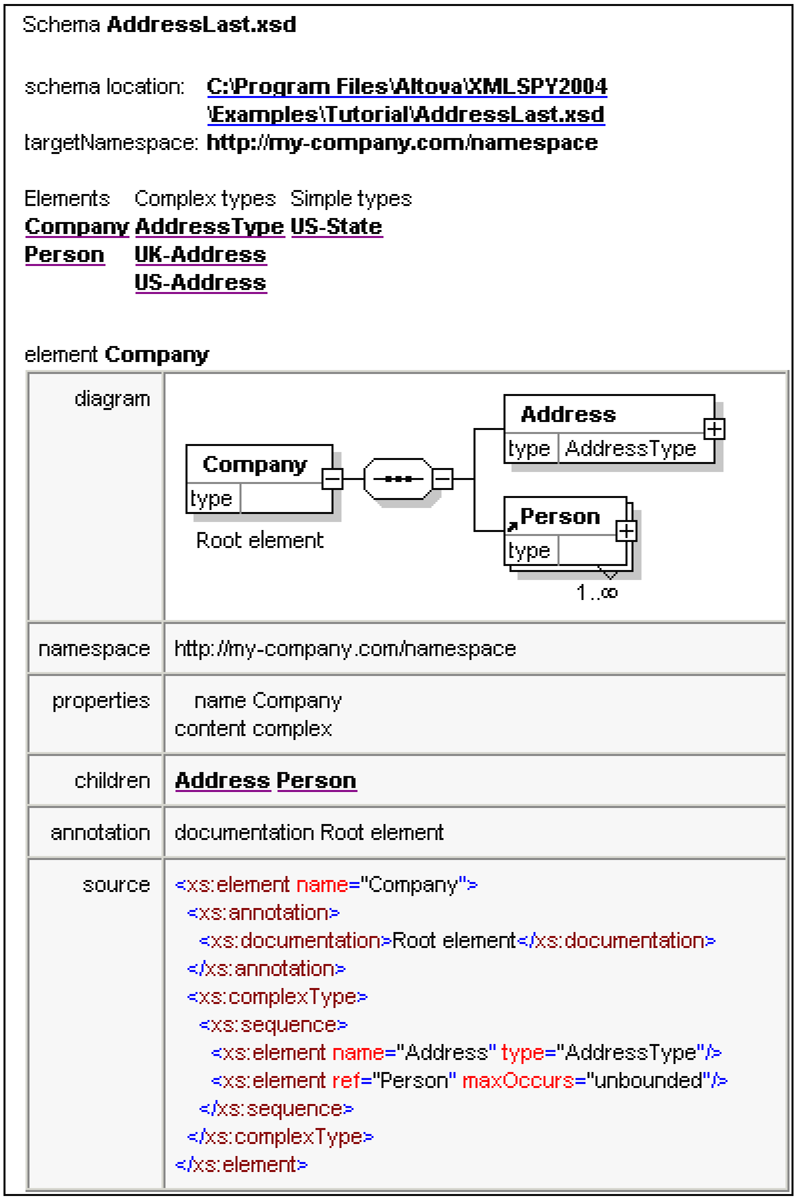
☑ Source code

☑ Show result file after export

☑ Save schema documentation settings

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\XMLSPY2004\Examples\Tutorial\AddressLast.xsd**
targetNamespace: **http://my-company.com/namespace**

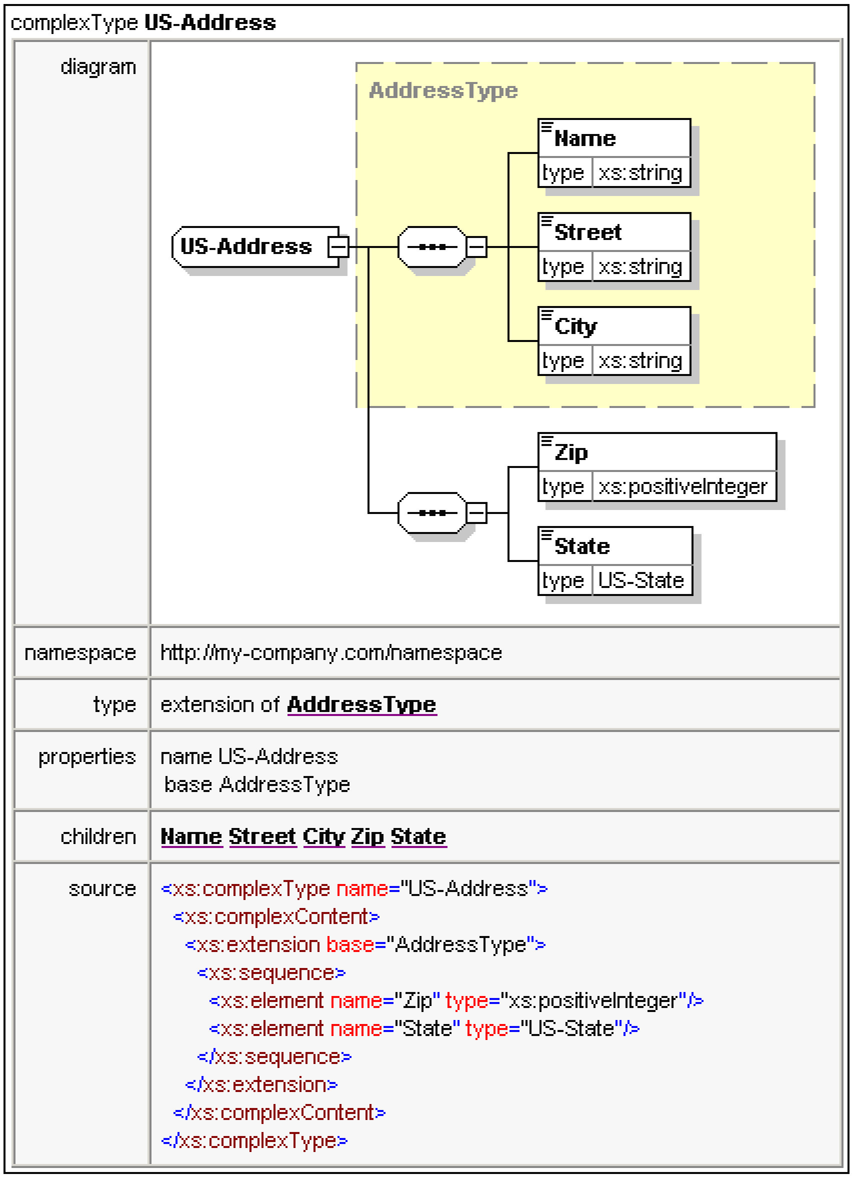
Elements	Complex types	Simple types
Company	**AddressType**	**US-State**
Person	**UK-Address**	
	US-Address	

element **Company**

diagram	
namespace	http://my-company.com/namespace
properties	name Company content complex
children	**Address** **Person**
annotation	documentation Root element
source	

```
<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>
```

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.

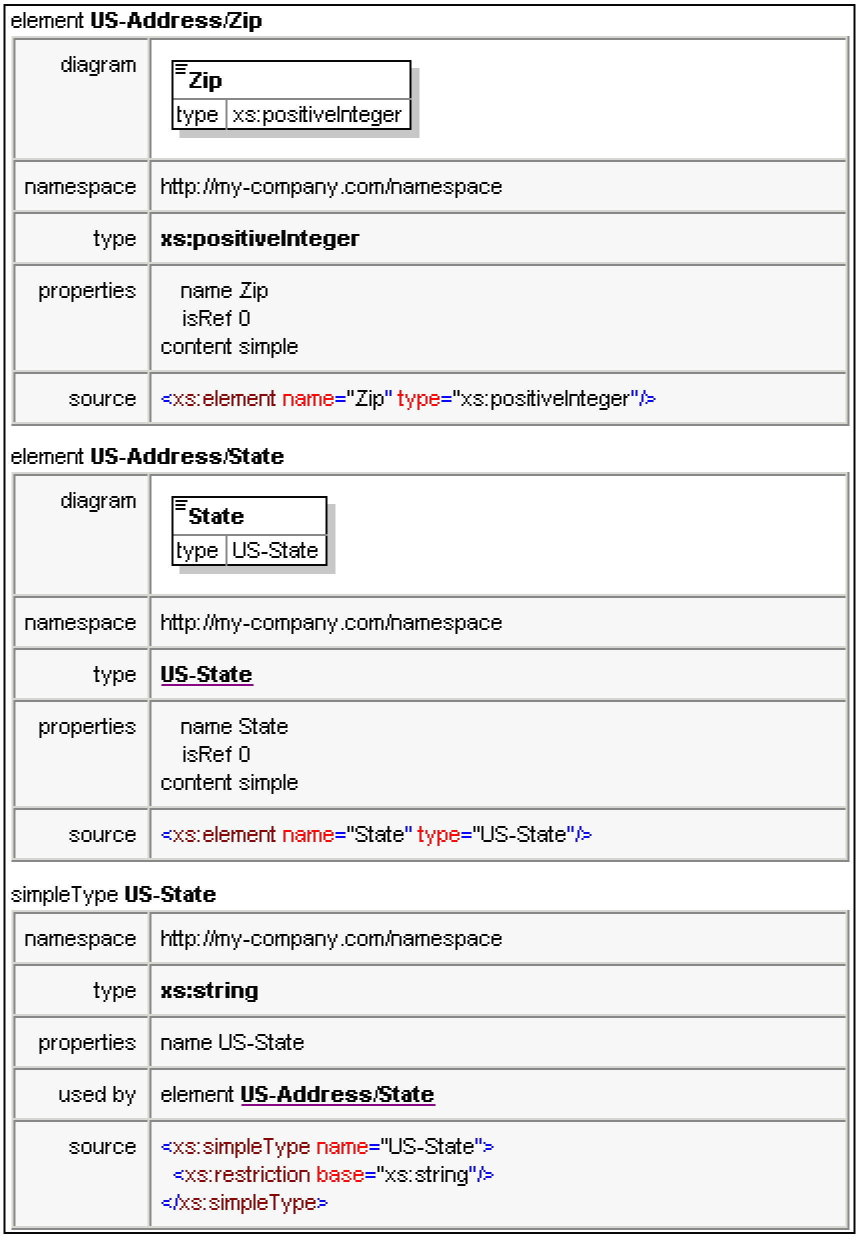


complexType **US-Address**

diagram	
namespace	http://my-company.com/namespace
type	extension of **AddressType**
properties	name US-Address base AddressType
children	**Name** **Street** **City** **Zip** **State**
source	

```
<xs:complexType name="US-Address">
  <xs:complexContent>
    <xs:extension base="AddressType">
      <xs:sequence>
        <xs:element name="Zip" type="xs:positiveInteger"/>
        <xs:element name="State" type="US-State"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

The diagram above shows how complex types are documented.



element **US-Address/Zip**

diagram	Zip type xs:positiveInteger
namespace	http://my-company.com/namespace
type	**xs:positiveInteger**
properties	name Zip isRef 0 content simple
source	<xs:element name="Zip" type="xs:positiveInteger"/>

element **US-Address/State**

diagram	State type US-State
namespace	http://my-company.com/namespace
type	**US-State**
properties	name State isRef 0 content simple
source	<xs:element name="State" type="US-State"/>

simpleType **US-State**

namespace	http://my-company.com/namespace
type	**xs:string**
properties	name US-State
used by	element **US-Address/State**
source	<xs:simpleType name="US-State"> <xs:restriction base="xs:string"/> </xs:simpleType>

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.

2.1.5 Creating an XML document

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
In this section of the tutorial you will 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 the Schema/WSDL View. The following commands are used:

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

View | Text View. Switches to Text View.

View | Enhanced 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 Enhanced Grid View.

F7. Checks for well-formedness.

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

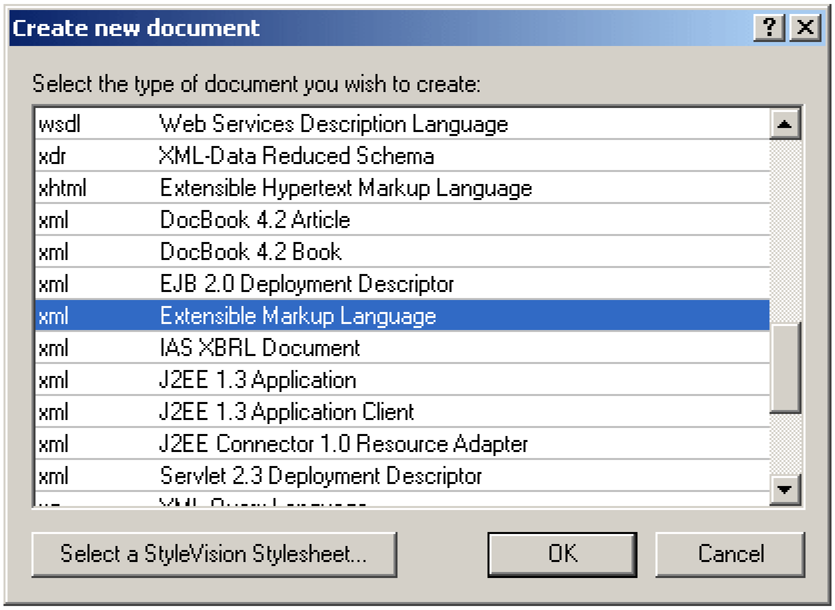
Opens the associated DTD or XML Schema file.

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:

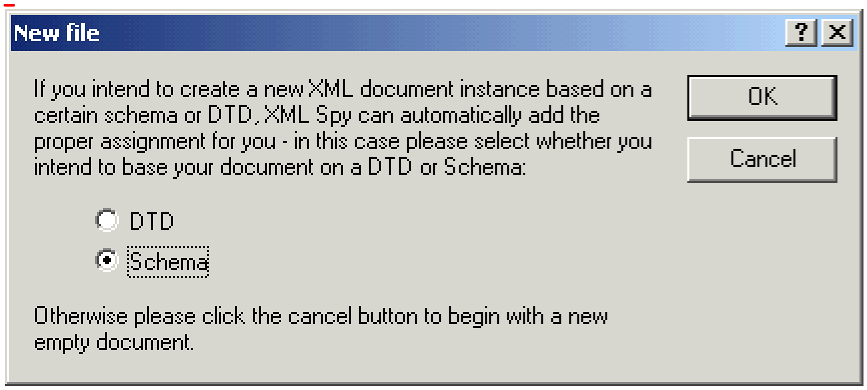
1. Select the menu option **File | New**. The Create new document dialog opens.



Select the type of document you wish to create:

wsdl	Web Services Description Language
xdr	XML-Data Reduced Schema
xhtml	Extensible Hypertext Markup Language
xml	DocBook 4.2 Article
xml	DocBook 4.2 Book
xml	EJB 2.0 Deployment Descriptor
xml	Extensible Markup Language
xml	IAS XBRL Document
xml	J2EE 1.3 Application
xml	J2EE 1.3 Application Client
xml	J2EE Connector 1.0 Resource Adapter
xml	Servlet 2.3 Deployment Descriptor
xq	XML Query Language

2. Select the `Extensible Markup Language` 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.



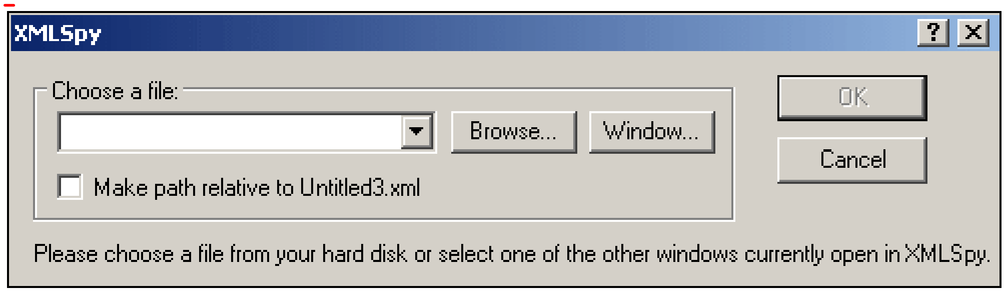
If you intend to create a new XML document instance based on a certain schema or DTD, XML Spy can automatically add the proper assignment for you - in this case please select whether you intend to base your document on a DTD or Schema:

○ DTD

● Schema

Otherwise please click the cancel button to begin with a new empty document.

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.



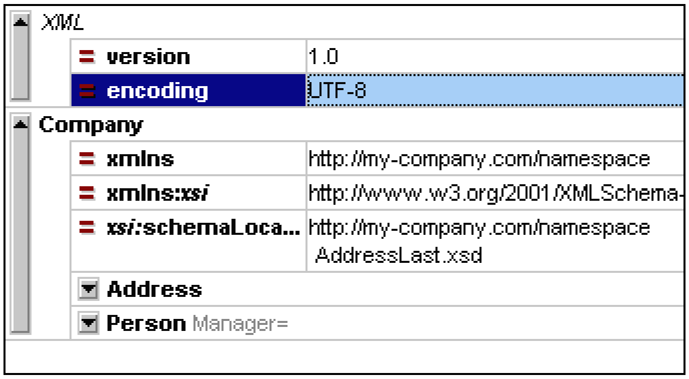
Choose a file:

□ Make path relative to Untitled3.xml

Please choose a file from your hard disk or select one of the other windows currently open in XMLSpy.

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`, 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 Enhanced Grid View.

6. In Grid View, the entire document is selected. Click on any element to reduce selection to that element. Your document should look something like this:



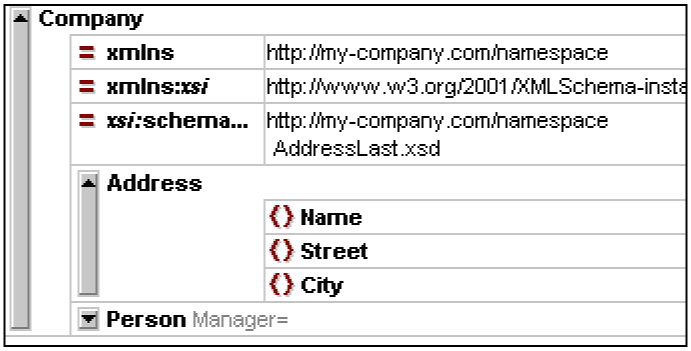
XML

= version	1.0
= encoding	UTF-8

Company

= xmlns	http://my-company.com/namespace
= xmlns:xsi	http://www.w3.org/2001/XMLSchema-
= xsi:schemaLoca...	http://my-company.com/namespace AddressLast.xsd
▼ Address	
▼ Person Manager=	

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



Company

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

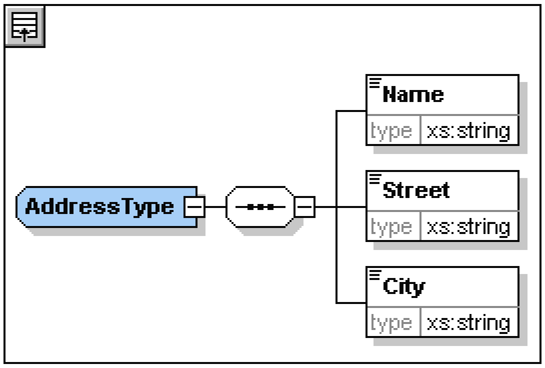
Address

() Name
() Street
() City

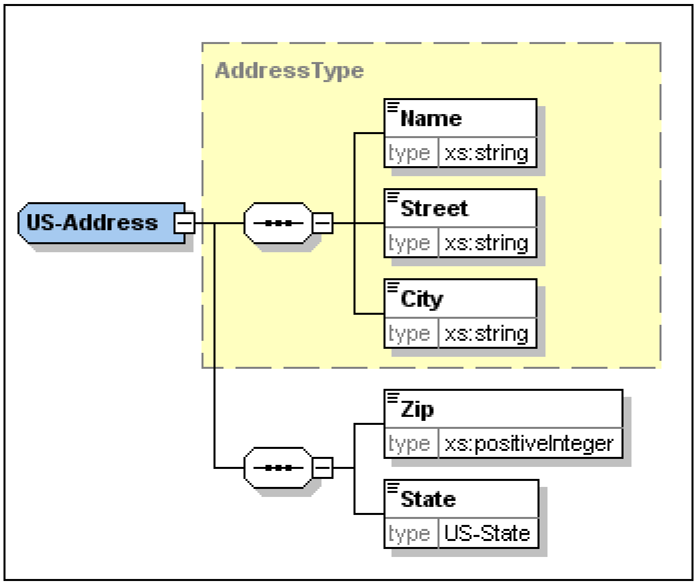
▼ Person Manager=

Specifying the type of an element

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



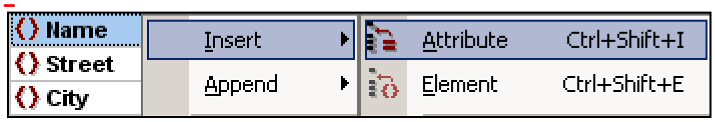
We would, however, like to use a specific US or UK address type rather than the generic address type. You will recall that, in the `AddressLast.xsd` schema, we created global complex types for `US-Address` and `UK-Address` by extending the `AddressType` complex type. The content model of `US-Address`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

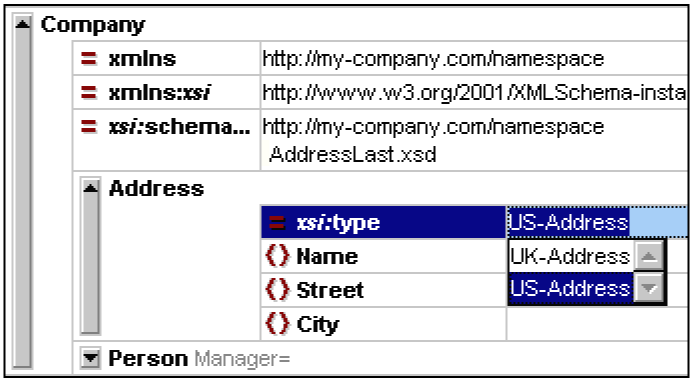
Do the following:

1. Right-click the `Name` element, and select **Insert | Attribute** from the context menu.



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

2. Enter `xsi:type` as the name of the attribute.
3. Press **Tab** to move into the next (value) field. A popup menu appears (*shown below*) listing the available complex types appears.


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

4. Select `US-Address` from the list, and confirm with **Enter**.

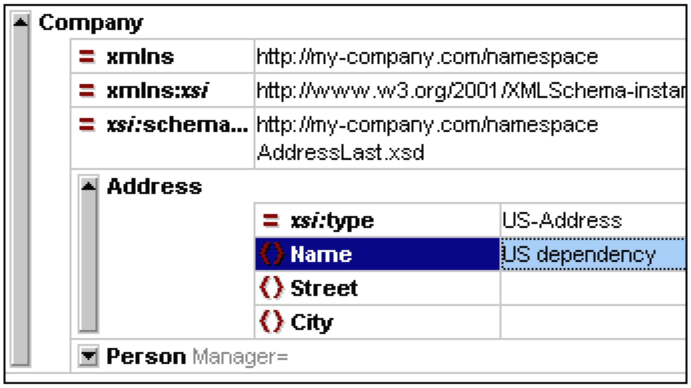
Please note: The `xsi` prefix allows you to use special XML Schema related commands in your XML document instance. In the above case, you have specified a type for the `Address` element. See the [XML Schema specification] for more information.

Entering data in Grid View

You can now enter data into your XML document.

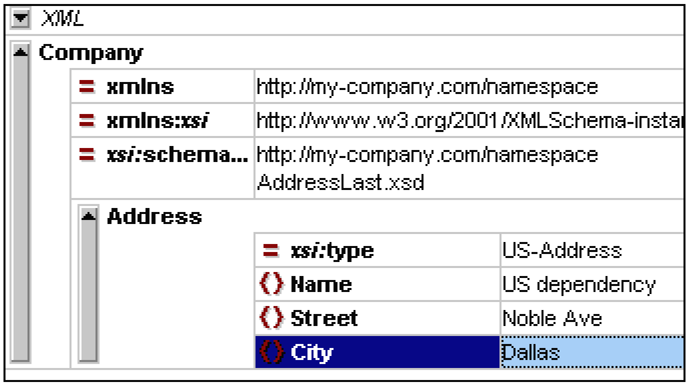
Do the following:

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



Company			
	= xmlns	http://my-company.com/namespace	
	= xmlns:*xsi*	http://www.w3.org/2001/XMLSchema-instan	
	= *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:



XML			
Company			
	= xmlns	http://my-company.com/namespace	
	= xmlns:*xsi*	http://www.w3.org/2001/XMLSchema-instan	
	= *xsi*:schema...	http://my-company.com/namespace AddressLast.xsd	
	Address		
		= *xsi*:type	US-Address
		{} Name	US dependency
		{} Street	Noble Ave
		{} City	Dallas

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 editing features

In addition, Text View provides a number of structural editing features that make editing large sections of text easy. Among these latter features are the following, which can be toggled on and off by clicking the appropriate icon.

Enables/disables line numbering.

Enables/disables the source folding margin.

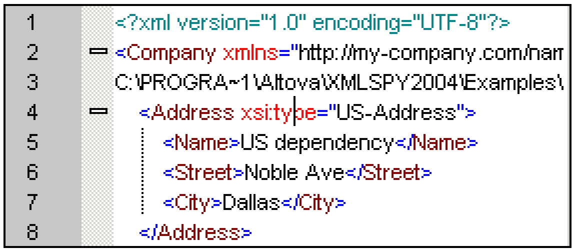
Enables/disables the bookmark margin.

Inserts/removes bookmarks.

Enables/disables indentation guides.

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.

Comment: The example file below needs a small correction. It says "US depencency" instead of "US dependency". The screenshot was edited to be correct, but the actual file has to be changed.



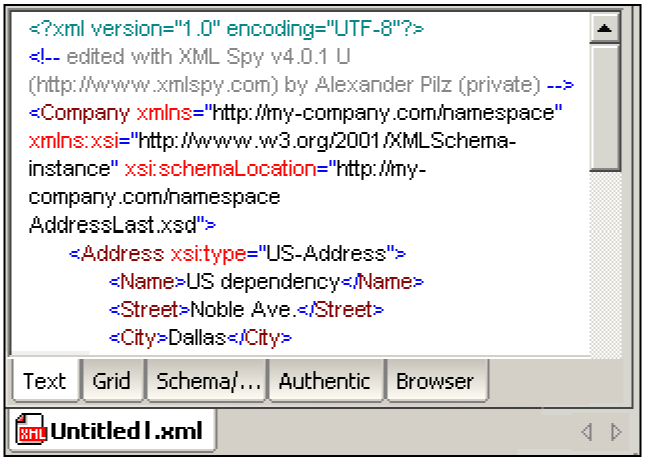
```
<?xml version="1.0" encoding="UTF-8"?>
<Company xmlns="http://my-company.com/nam
C:\PROGRA~1\Altova\XMLSPY2004\Examples\
  <Address xsi:type="US-Address">
    <Name>US dependency</Name>
    <Street>Noble Ave</Street>
    <City>Dallas</City>
  </Address>
```

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.

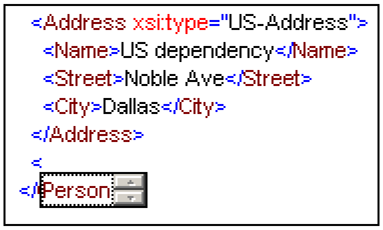
Do the following:

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.



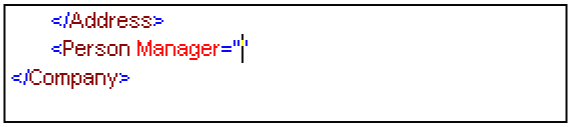
```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.0.1 U
(http://www.xmlspy.com) by Alexander Pilz (private) -->
<Company xmlns="http://my-company.com/namespace"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://my-
company.com/namespace
AddressLast.xsd">
    <Address xsi:type="US-Address">
        <Name>US dependency</Name>
        <Street>Noble Ave.</Street>
        <City>Dallas</City>
```

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.



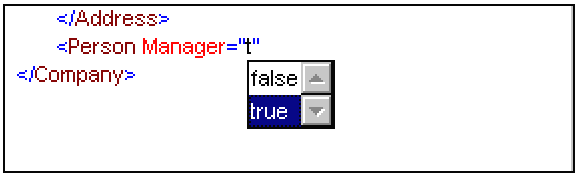
```
<Address xsi:type="US-Address">
  <Name>US dependency</Name>
  <Street>Noble Ave</Street>
  <City>Dallas</City>
</Address>
<
</Person
```

4. Select the `Person` entry. The `Person` element, as well as its attribute `Manager`, are inserted.



```
    </Address>
    <Person Manager=""
</Company>
```

5. Enter the letter `"t"` within the quotes. This opens a dropdown list where `true` is highlighted.

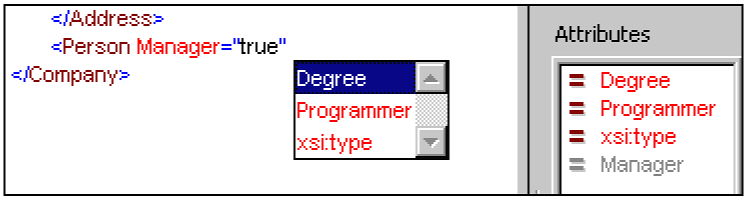


```
    </Address>
    <Person Manager="t"
</Company>
```

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

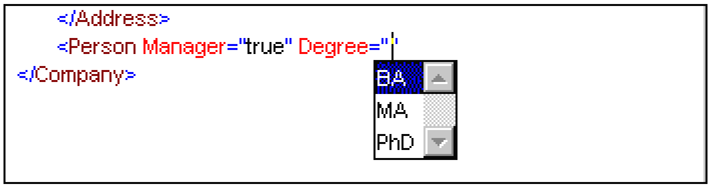
6. Move the cursor to the end of the line (using the **End** key if you like), and press the

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.



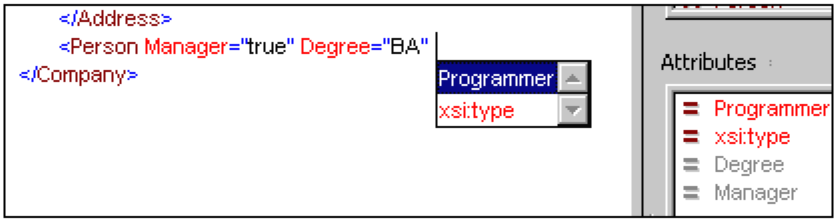
```
    </Address>
    <Person Manager="true"
</Company>
```

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`).



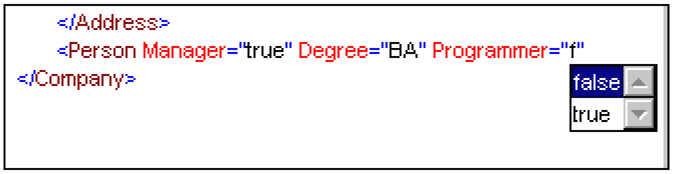
```
    </Address>
    <Person Manager="true" Degree=""
</Company>
```

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.



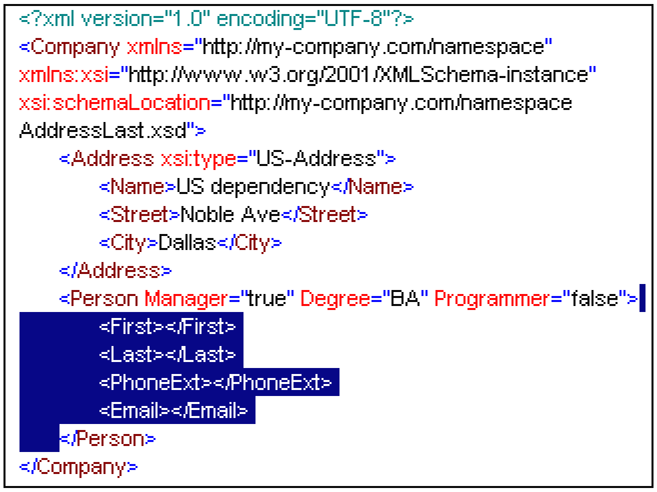
```
    </Address>
    <Person Manager="true" Degree="BA"
</Company>
```

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



```
    </Address>
    <Person Manager="true" Degree="BA" Programmer="f"
</Company>
```

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.

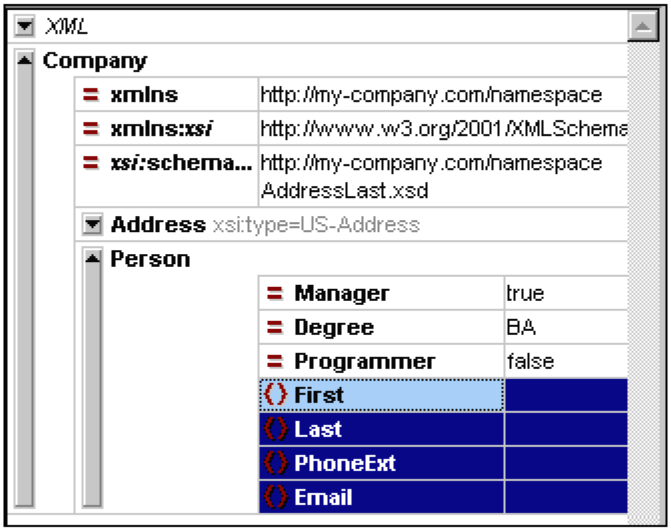


```
<?xml version="1.0" encoding="UTF-8"?>
<Company xmlns="http://my-company.com/namespace"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://my-company.com/namespace
AddressLast.xsd">
    <Address xsi:type="US-Address">
        <Name>US dependency</Name>
        <Street>Noble 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 | Enhanced Grid View,** or click the **Grid** tab. The newly added child elements of `Person` are highlighted.



XML

Company

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

Address xsi:type=US-Address

Person

= Manager	true
= Degree	BA
= Programmer	false
() First	
() Last	
() PhoneExt	
() Email	

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

Validating the document

XMLSpy provides two evaluations of the XML document:

- A well-formedness check
- A validation check

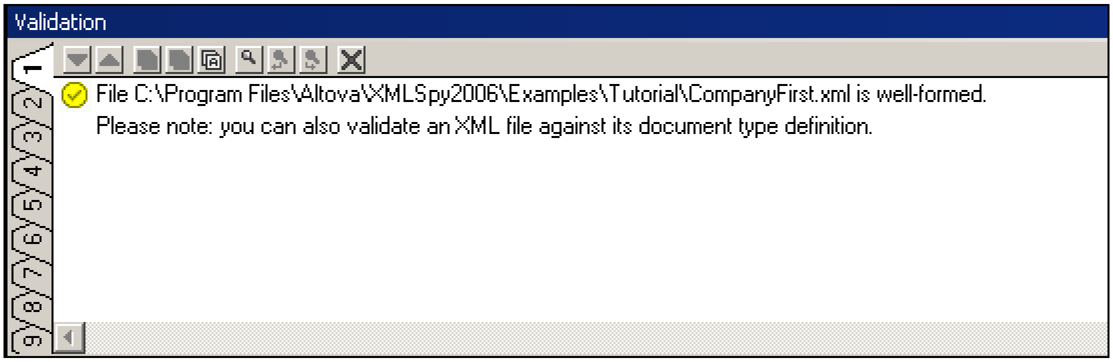
If either of these checks fails, we will have to modify the document appropriately.

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.
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 Validation window at the bottom of the Main Window saying the document is well-formed.

Notice that 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).



File C:\Program Files\Altova\XMLSpy2006\Examples\Tutorial\CompanyFirst.xml is well-formed.
Please note: you can also validate an XML file against its document type definition.

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.

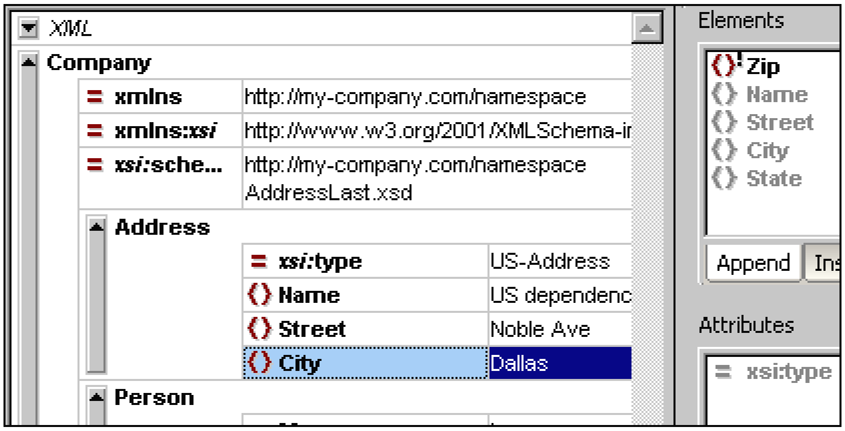
Checking validity

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, select the menu option **XML | Validate**, or press the **F8** key, or click . An error message appears in the Validation 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.



XML

Company

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

Address

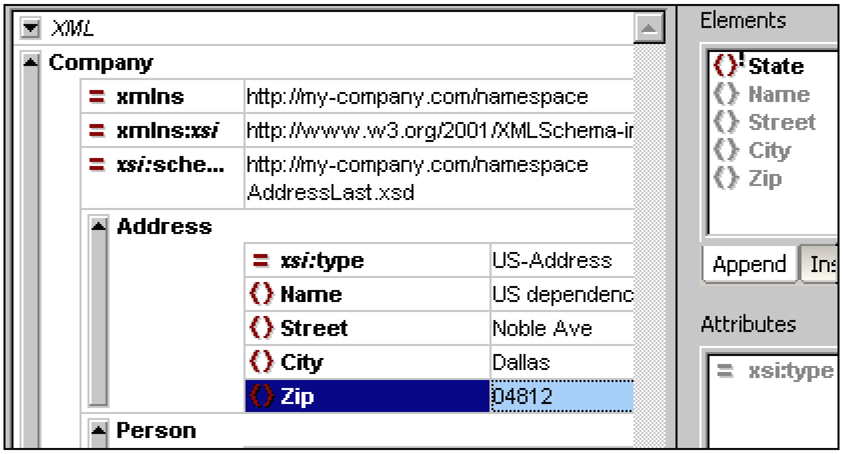
= xsi:type	US-Address
() Name	US dependenc
() Street	Noble Ave
() City	Dallas

Person

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`), and 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*.



XML

Company

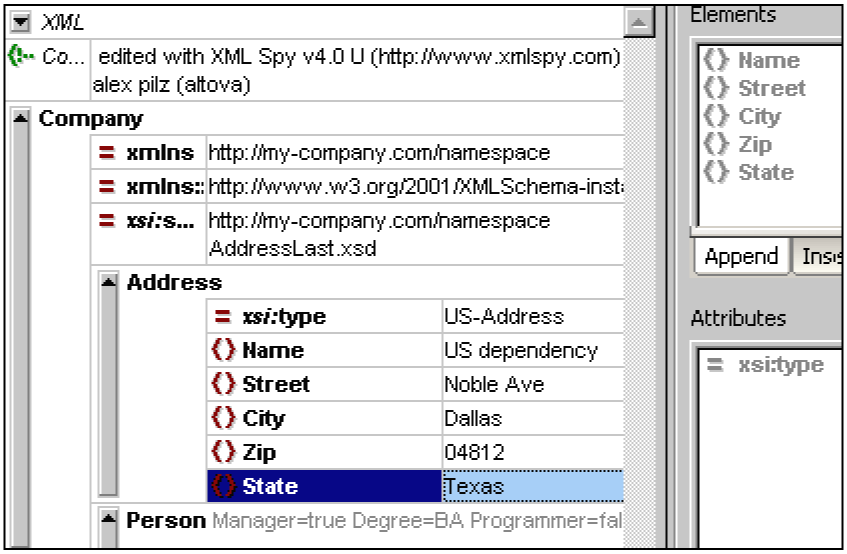
= xmlns	http://my-company.com/namespace
= xmlns:xsi	http://www.w3.org/2001/XMLSchema-ir
= xsi:sche...	http://my-company.com/namespace AddressLast.xsd

Address

= xsi:type	US-Address
() Name	US dependenc
() Street	Noble Ave
() City	Dallas
() Zip	04812

Person

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`.



XML

Co...	edited with XML Spy v4.0 U (http://www.xmlspy.com) alex pilz (altova)

Company

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

Address

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

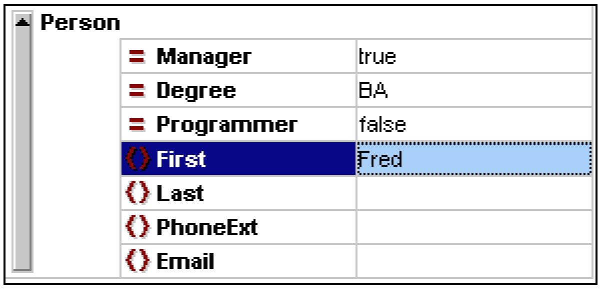
Person Manager=true Degree=BA Programmer=fal

Completing the document and revalidating

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

Do the following:

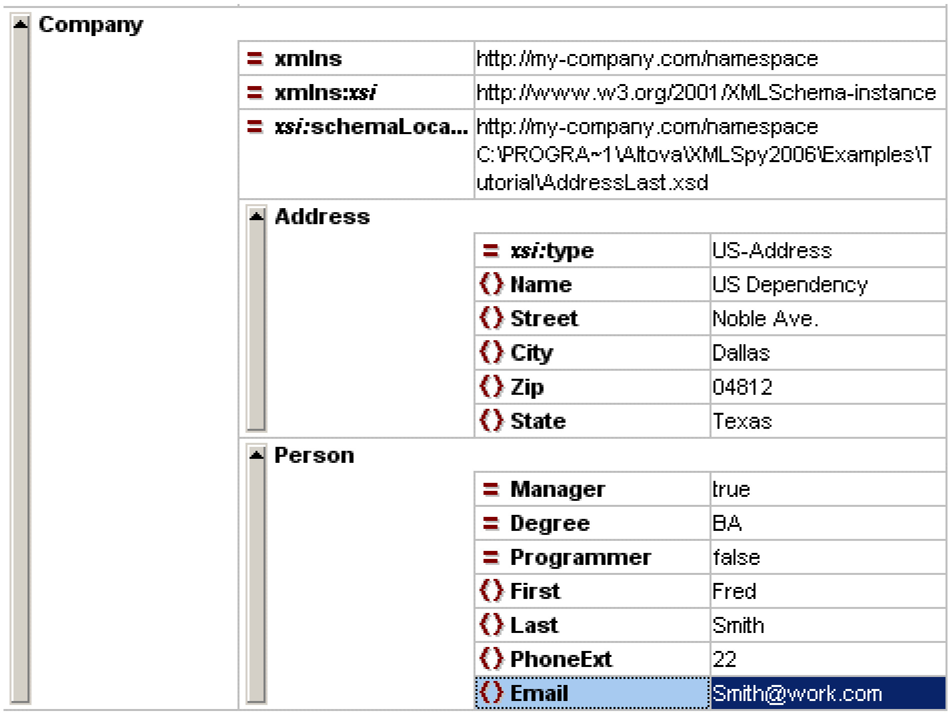
1. Click the value field of the element `First`, and enter a first name (say `Fred`). Then press **Enter**.



Person

= Manager	true
= Degree	BA
= Programmer	false
() First	Fred
() Last	
() PhoneExt	
() Email	

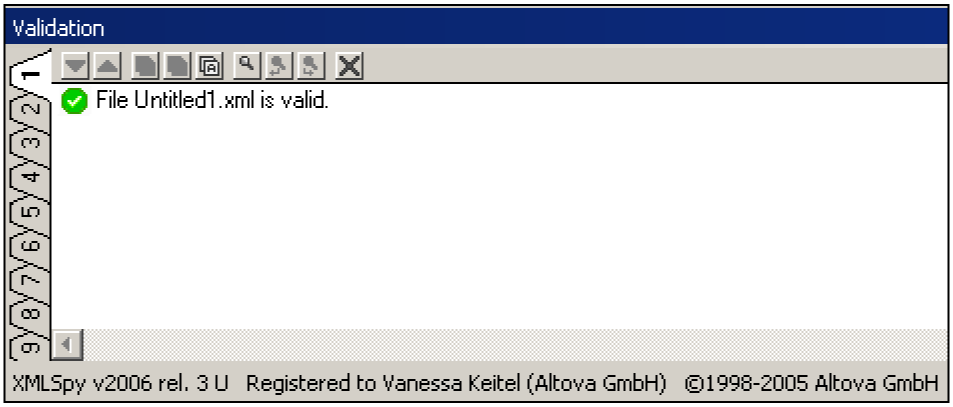
2. In the same way enter data for all the child elements of `Person`, that is, for `Last`, `PhoneExt`, and `Email`. Note that the value of `PhoneExt` must be an integer with a maximum value of 99 (since this is the range of allowed `PhoneExt` values you defined in your schema). Your XML document should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



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 Validation window stating that the file is valid. The XML document is now valid against its schema.



✓ File Untitled1.xml is valid.

XMLSpy v2006 rel. 3 U Registered to Vanessa Keitel (Altova GmbH) ©1998-2005 Altova GmbH

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.

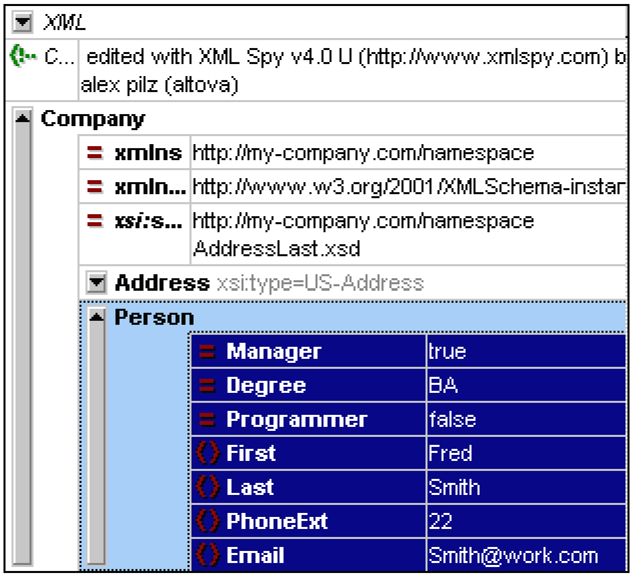
Please note: An XML document does not have to be valid in order to save it. Saving an invalid document causes a prompt to appear warning you that you are about to save an invalid document. You can select **Save anyway**, if you wish to save the document in its current invalid state.

Appending elements and attributes in Grid View

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.



XML

C... edited with XML Spy v4.0 U (http://www.xmlspy.com) b alex pilz (altova)

Company

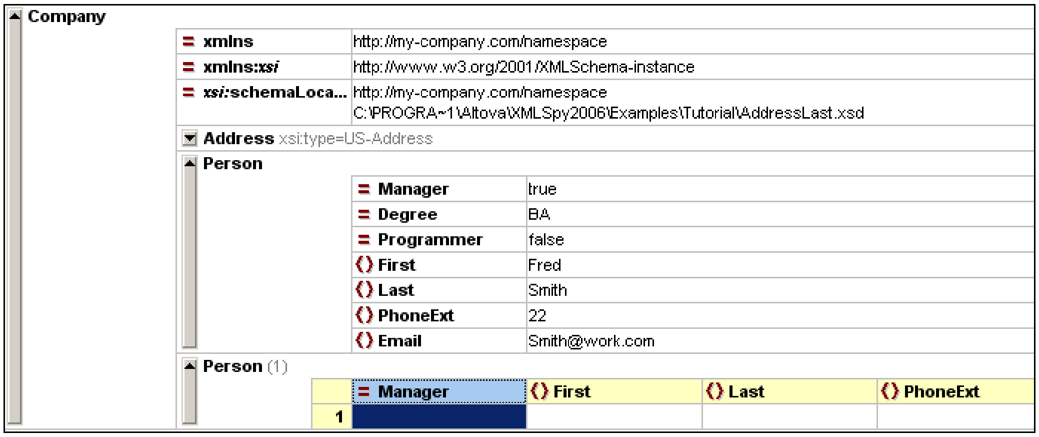
= xmlns	http://my-company.com/namespace
= xmln...	http://www.w3.org/2001/XMLSchema-instar
= xsi:s...	http://my-company.com/namespace AddressLast.xsd

Address xsi:type=US-Address

Person

= Manager	true
= Degree	BA
= Programmer	false
{} First	Fred
{} Last	Smith
{} PhoneExt	22
{} Email	Smith@work.com

3. 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.



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

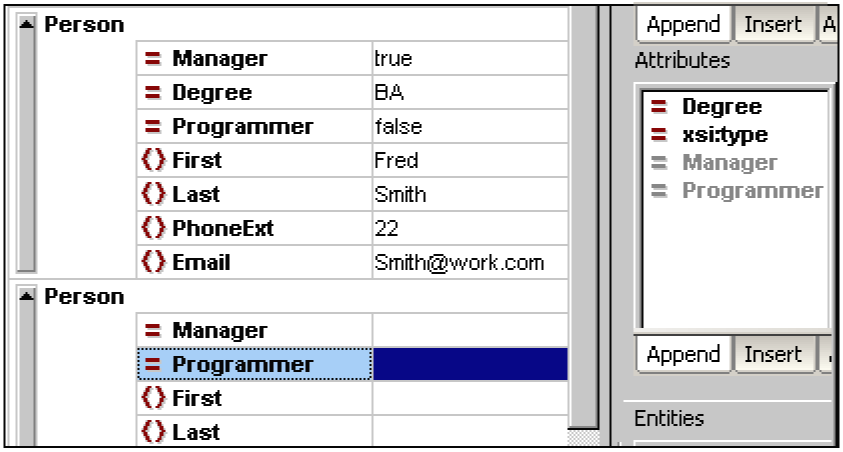
Person

= Manager	true
= Degree	BA
= Programmer	false
{} First	Fred
{} Last	Smith
{} PhoneExt	22
{} Email	Smith@work.com

Person (1)

	= Manager	{} First	{} Last	{} PhoneExt
1				

4. Press **F12** to switch the new `Person` element to Grid View.
5. 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.
6. In the **Append** tab of the Attributes Entry Helper, double-click the `Programmer` entry. This inserts an empty `Programmer` attribute after the `Manager` attribute.



Person

= Manager	true
= Degree	BA
= Programmer	false
{} First	Fred
{} Last	Smith
{} PhoneExt	22
{} Email	Smith@work.com

Person

= Manager	
= Programmer	
{} First	
{} Last	

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`.

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

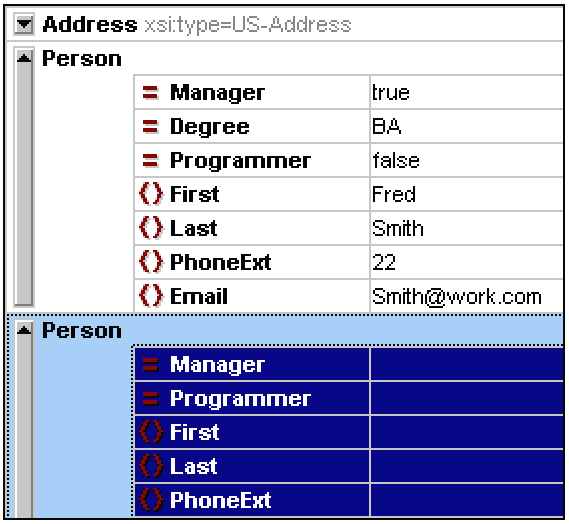
Table View provides the following advantages:

- 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:

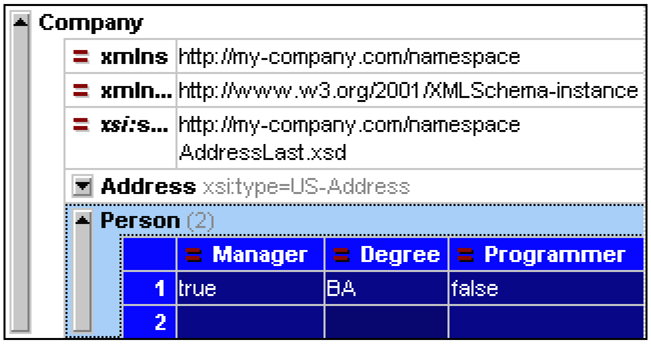
1. In Grid View, select either of the `Person` elements by clicking on or near the `Person` text.



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.



Company

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

Address xsi:type=US-Address

Person (2)

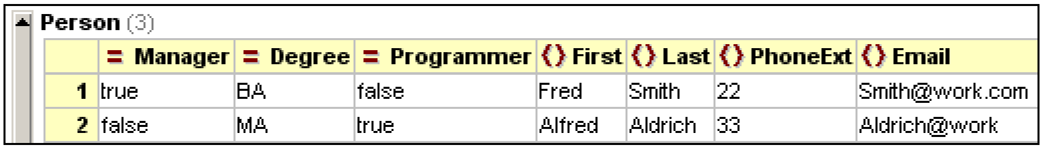
	= Manager	= Degree	= Programmer
1	true	BA	false
2			

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).



Person (3)

	= 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

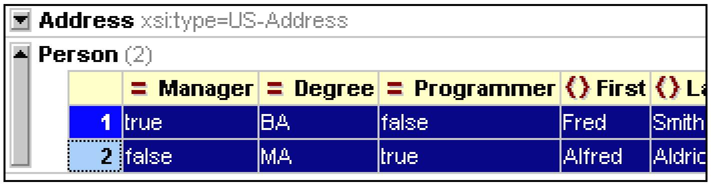
Please note: The Entry Helpers are active also for the elements and attributes displayed as a table. Double-clicking the `Person` entry in the Elements Entry Helper, for example, would add a new row to the table (i.e., a new occurrence of the `Person` element).

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.

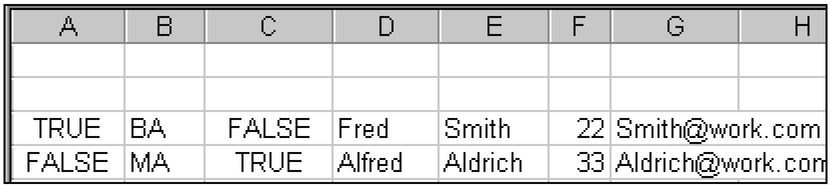


Address xsi:type=US-Address

Person (2)

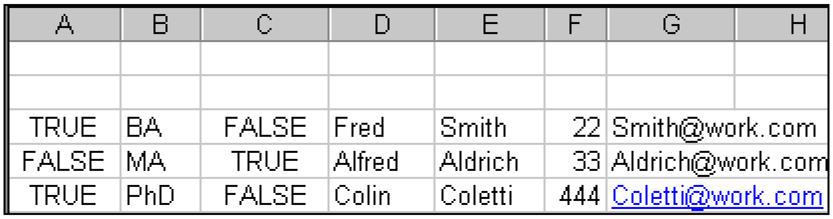
	= Manager	= Degree	= Programmer	() First	() La
1	true	BA	false	Fred	Smith
2	false	MA	true	Alfred	Aldric

2. Select the menu option **Edit | Copy as Structured text**. This command copies elements to the clipboard as they appear on screen.
3. Switch to Excel and paste (**Ctrl+V**) the XML data in an Excel worksheet.



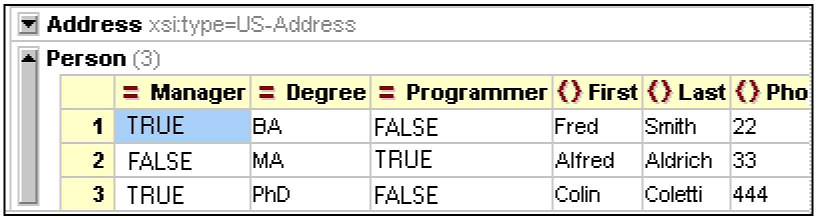
A	B	C	D	E	F	G	H
TRUE	BA	FALSE	Fred	Smith	22	Smith@work.com	
FALSE	MA	TRUE	Alfred	Aldrich	33	Aldrich@work.com	

4. Enter a new row of data in Excel. Make sure that you enter a three digit number for the `PhoneExt` element (say, `444`).



A	B	C	D	E	F	G	H
TRUE	BA	FALSE	Fred	Smith	22	Smith@work.com	
FALSE	MA	TRUE	Alfred	Aldrich	33	Aldrich@work.com	
TRUE	PhD	FALSE	Colin	Coletti	444	Coletti@work.com	

5. Mark the table data in Excel, and select **Edit | Copy** to copy the data to the clipboard.
6. Switch back to XMLSpy.
7. Click in the top left **data** cell of the table in XMLSpy, and select **Edit | Paste**.



Address xsi:type=US-Address

Person (3)

	= Manager	= Degree	= Programmer	() First	() Last	() Pho
1	TRUE	BA	FALSE	Fred	Smith	22
2	FALSE	MA	TRUE	Alfred	Aldrich	33
3	TRUE	PhD	FALSE	Colin	Coletti	444

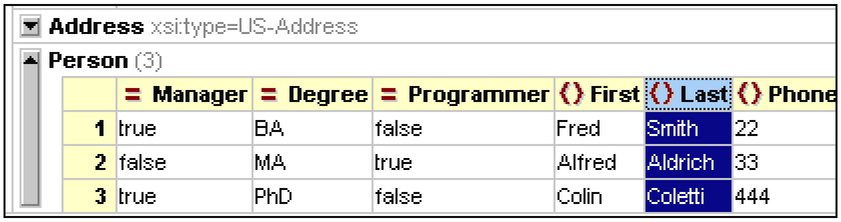
8. The updated table data is now visible in the table.
9. Change the uppercase boolean values `TRUE` and `FALSE` to lowercase `true` and `false` , respectively, using the menu option **Edit | Replace (Ctrl+H)**.

Sorting the table by the contents of a column

A table in Table View can be sorted in ascending or descending order by any of its columns. In this case, we want to sort the `Person` table by last names.

To sort a table by the contents of a column:

1. Select the `Last` column by clicking in its header.

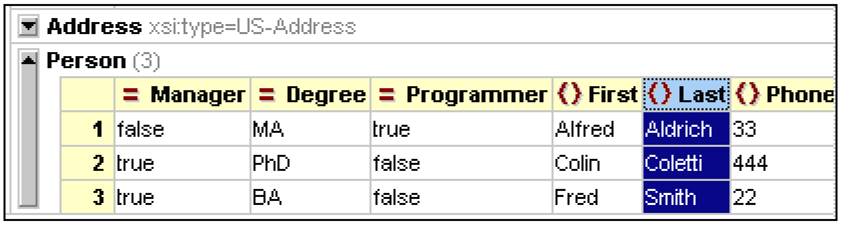


Address xsi:type=US-Address

Person (3)

	= 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.



Address xsi:type=US-Address

Person (3)

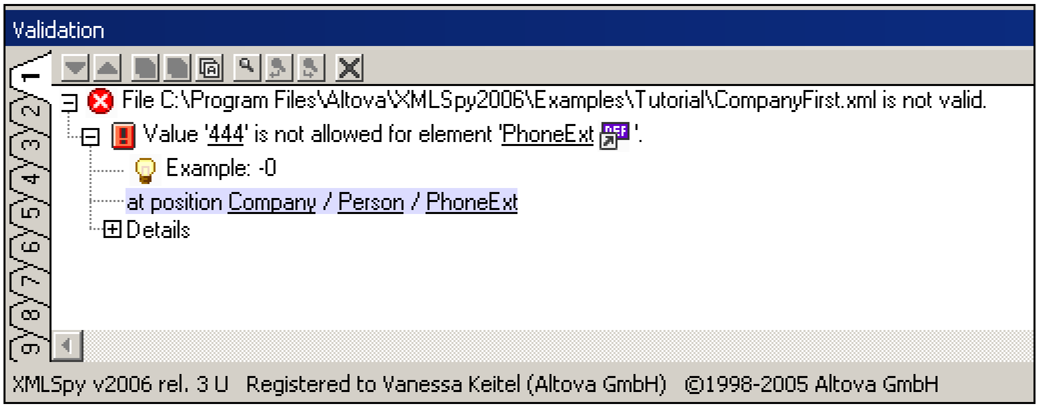
	= Manager	= Degree	= Programmer	() First	() Last	() Phone
1	false	MA	true	Alfred	Aldrich	33
2	true	PhD	false	Colin	Coletti	444
3	true	BA	false	Fred	Smith	22

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.



Validation

File C:\Program Files\Altova\XMLSpy2006\Examples\Tutorial\CompanyFirst.xml is not valid.

Value '444' is not allowed for element 'PhoneExt '.

Example: -0

at position Company / Person / PhoneExt

Details

XMLSpy v2006 rel. 3 U Registered to Vanessa Keitel (Altova GmbH) ©1998-2005 Altova GmbH

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.

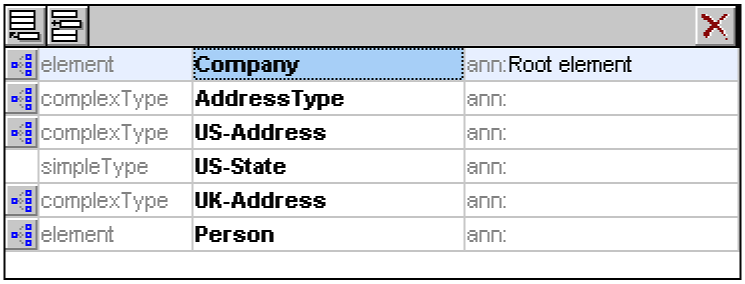
Modifying the schema

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.

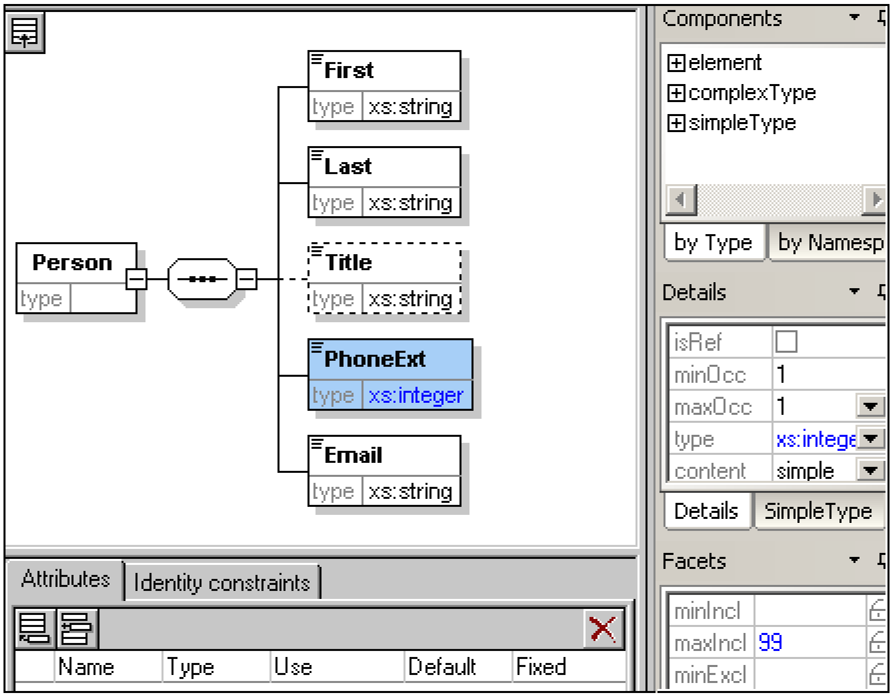
Do the following:

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`. Switch to Schema/WSDL View (screenshot below).



element	**Company**	ann:Root element
complexType	**AddressType**	ann:
complexType	**US-Address**	ann:
simpleType	**US-State**	ann:
complexType	**UK-Address**	ann:
element	**Person**	ann:

2. 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.



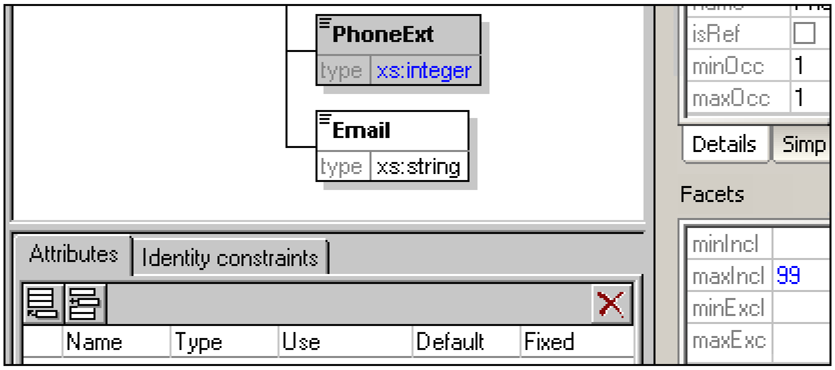
Details

isRef	□
minOcc	1
maxOcc	1
type	xs:intege
content	simple

Facets

minIncl	
maxIncl	99
minExcl	

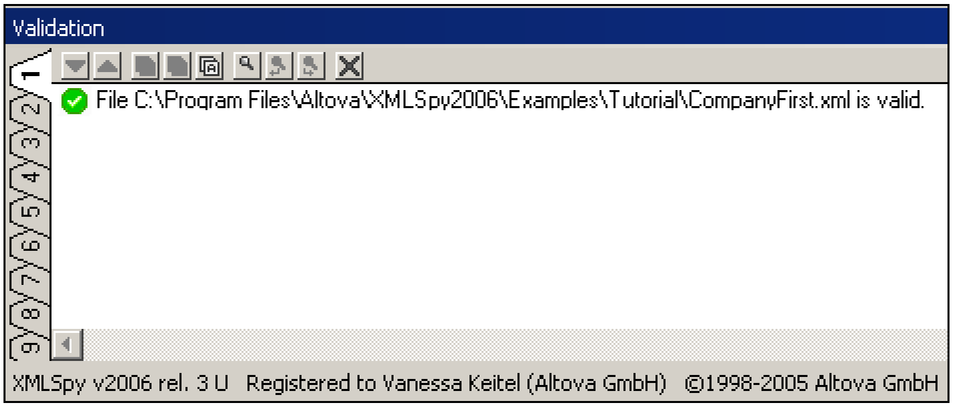
3. In the Facets tab, double-click the `maxIncl` value field, change the value `99` to `999`, and confirm with **Enter**.



isRef	☐
minOcc	1
maxOcc	1

minIncl	
maxIncl	99
minExcl	
maxExc	

4. Save the schema document.
5. Press **Ctrl+Tab** to switch back to the XML document.
6. Click to revalidate the XML document.



File C:\Program Files\Altova\XMLSpy2006\Examples\Tutorial\CompanyFirst.xml is valid.

XMLSpy v2006 rel. 3 U Registered to Vanessa Keitel (Altova GmbH) ©1998-2005 Altova GmbH

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.

2.1.6 Using XSLT to transform XML

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.

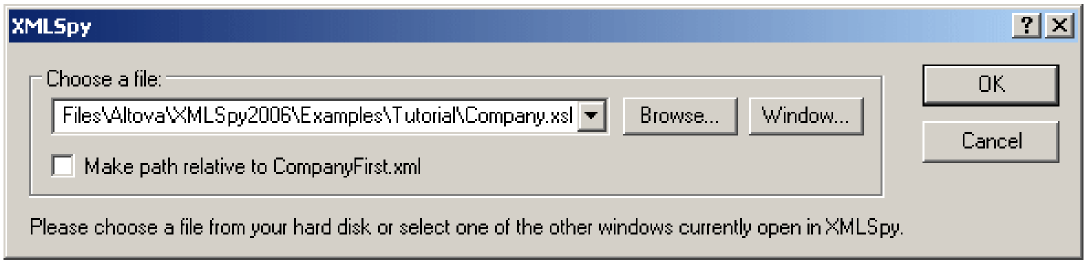
XSL/XQuery | XSL Transformation (F10), or the toolbar icon , transforms the active XML document using the XSL stylesheet assigned to the XML file. If an XSL file has not been assigned then you will be prompted for one when you select this command.

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.

Assigning an XSL file

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

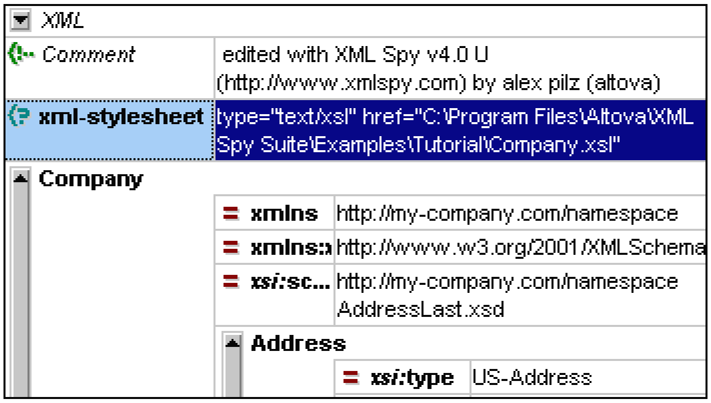
1. Click the `CompanyLast.xml` tab in the main window so that `CompanyLast.xml` becomes the active document.
2. Select the menu command **XSL/XQuery | Assign XSL**.
3. Click the **Browse** button, and select the `Company.xsl` file from the Tutorial folder. In the dialog, you can activate the option Make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



Choose a file:
Files\Altova\XMLSpy2006\Examples\Tutorial\Company.xsl
☐ Make path relative to CompanyFirst.xml

Please choose a file from your hard disk or select one of the other windows currently open in XMLSpy.

4. Click **OK** to assign the XSL file to the XML document.



XML
Comment: edited with XML Spy v4.0 U (http://www.xmlspy.com) by alex pilz (altova)
xml-stylesheet: type="text/xsl" href="C:\Program Files\Altova\XML Spy Suite\Examples\Tutorial\Company.xsl"
Company
= xmlns: http://my-company.com/namespace
= xmlns:x http://www.w3.org/2001/XMLSchema
= xsi:sc... http://my-company.com/namespace AddressLast.xsd
Address
= xsi:type US-Address

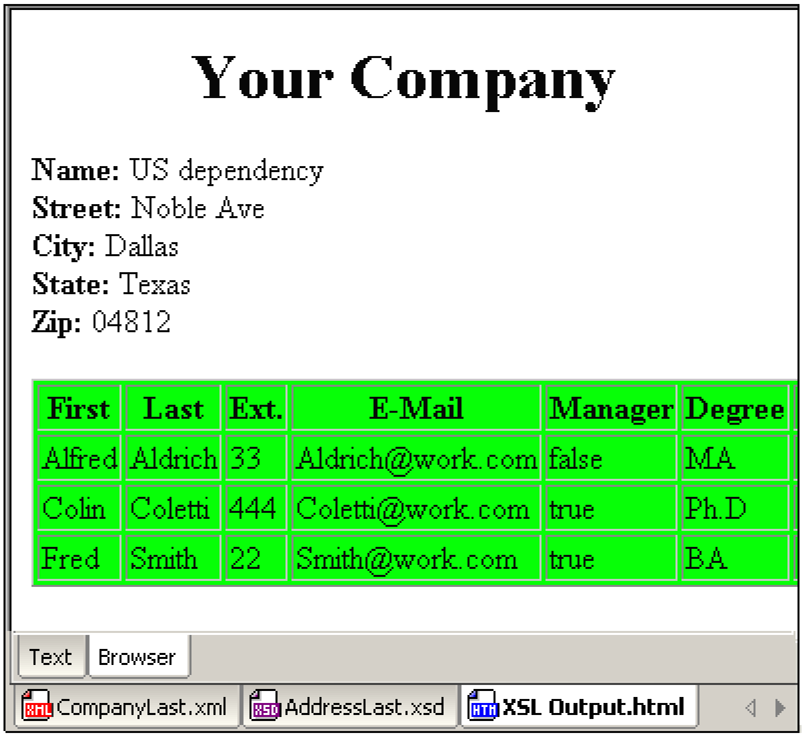
An `XML-stylesheet` processing instruction is inserted in the XML document that references the XSL file. If you activated the Make Path Relative to CompanyLast.xml check box, then the path is relative; otherwise absolute (as in the screenshot above).

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`. It shows the Company data in one block down the left, and the Person data in tabular form below.



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

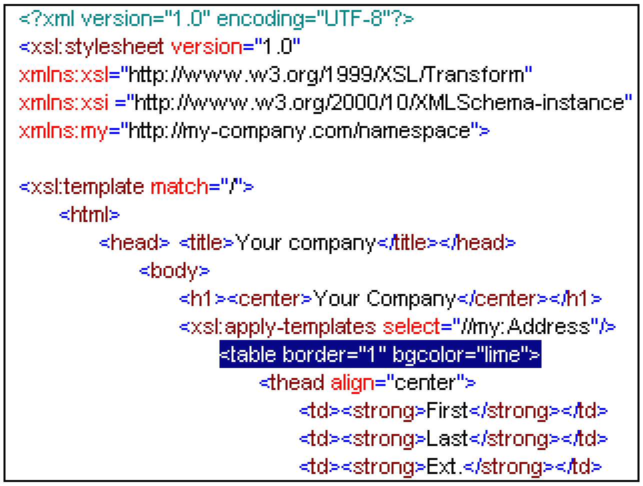
Please note: Should you only see a table header and no table data in the output file, make sure that you have defined the target namespace for your schema as detailed in Defining your own namespace at the beginning of the tutorial. The namespace must be **identical** in all three files (Schema, XML,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

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:

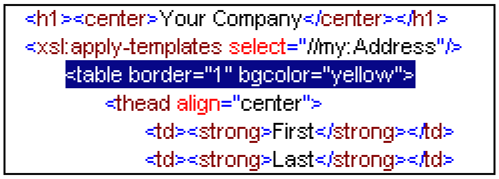
1. Click the `CompanyLast.xml` tab to make it the active document, and make sure you are in Grid View.
2. Select the menu option **XSL/XQuery | Go to XSL**.



```
<?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>
```

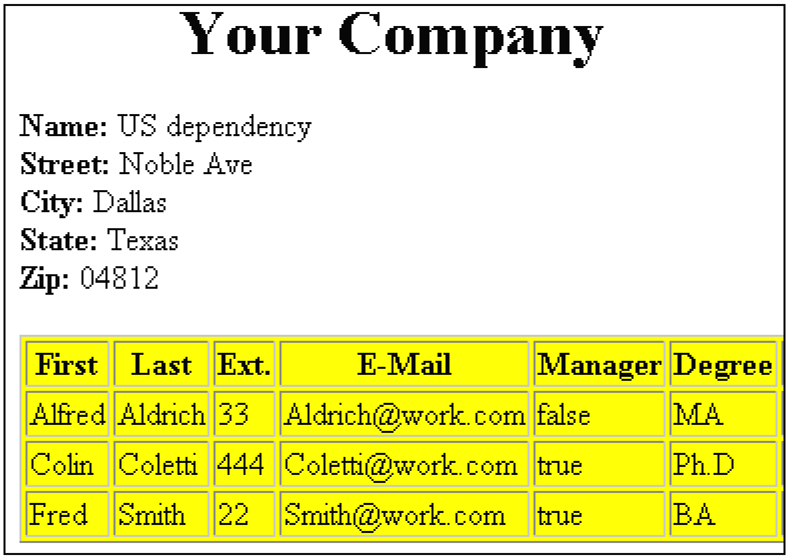
The command opens the `Company.xsl` file referenced in the XML document.

3. 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>
```

4. Select the menu option **File | Save** to save the changes made to the XSL file.
5. 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

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

2.1.7 Working with databases

Objective

To export Person data from our address list to MS Access 2000 pr MS Access 2003, and reimport the Person table into XMLSpy.

This will be achieved by:

- using the menu option **Convert**, and then selecting the export or import process.

Commands used in this section
Functions in this section:

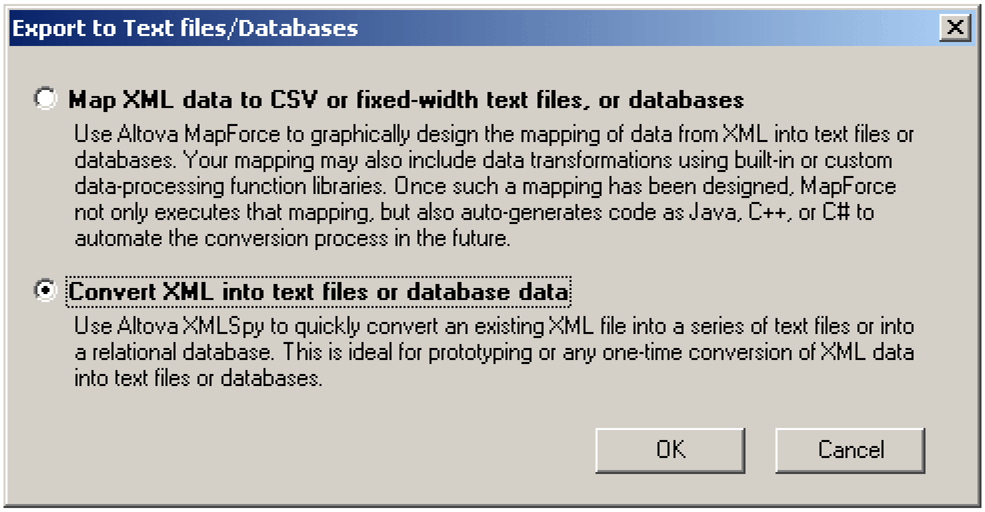
Convert | Export to Text files / Database, enables you to export XML data as text or for use in third party databases.

Convert | Import Database data, enables you to import database data into XMLSpy.

Exporting XML data to external databases

To export data to a database:

1. Click the **CompanyLast.xml** tab on the main window, to make it the active document.
2. Select the menu option **Convert | Export to Text files / Database...**. The Export to Text files/Databases dialog appears.

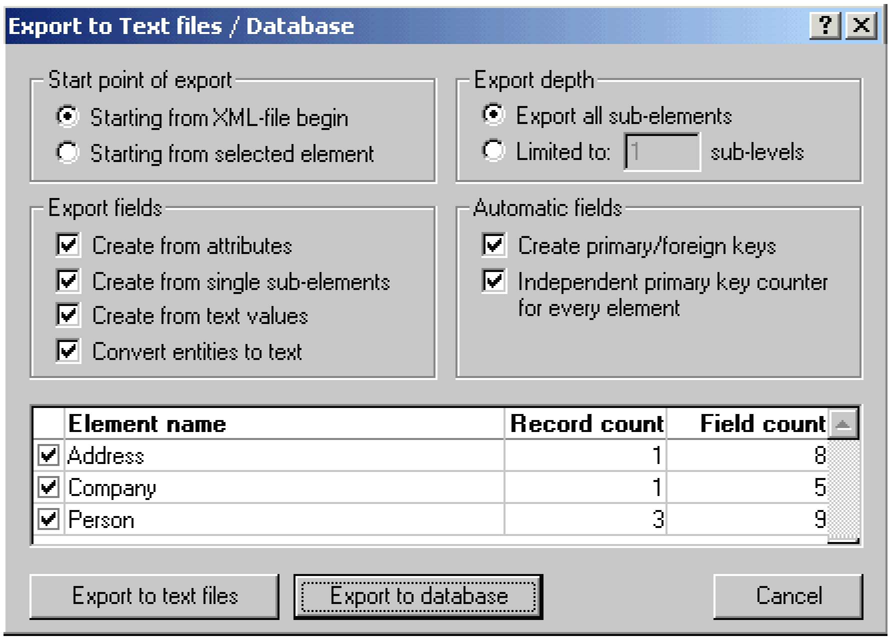


Export to Text files/Databases

○ **Map XML data to CSV or fixed-width text files, or databases**
Use Altova MapForce to graphically design the mapping of data from XML into text files or databases. Your mapping may also include data transformations using built-in or custom data-processing function libraries. Once such a mapping has been designed, MapForce not only executes that mapping, but also auto-generates code as Java, C++, or C# to automate the conversion process in the future.

● **Convert XML into text files or database data**
Use Altova XMLSpy to quickly convert an existing XML file into a series of text files or into a relational database. This is ideal for prototyping or any one-time conversion of XML data into text files or databases.

3. Click the **Convert XML into text files or database data** radio button, and confirm with **OK**. The Export to Text files/Database dialog appears.
The default settings in this dialog export all elements, attributes, and generate primary and foreign keys.



Export to Text files / Database

Start point of export
- ● Starting from XML-file begin
- ○ Starting from selected element

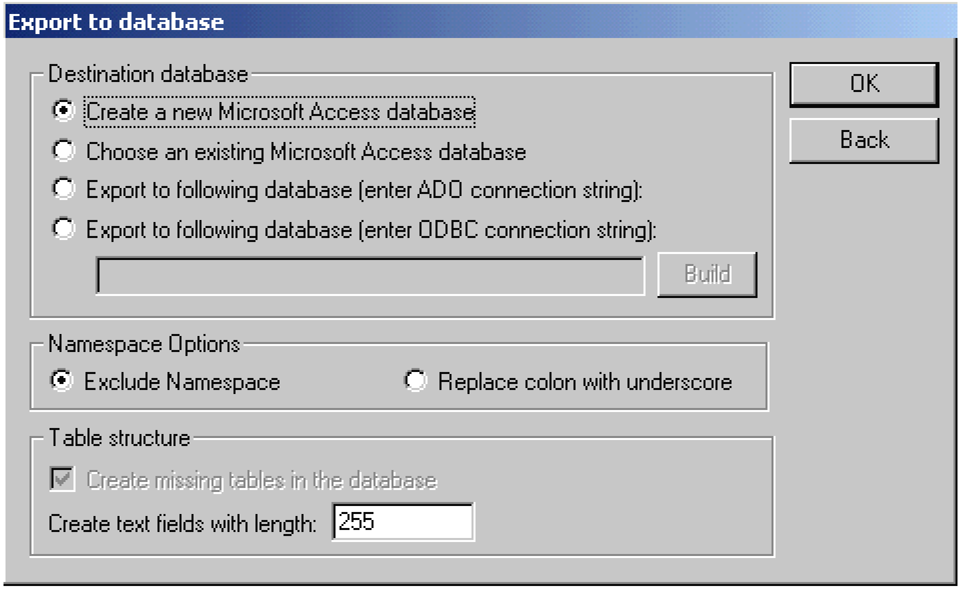
Export depth
- ● Export all sub-elements
- ○ Limited to: 1 sub-levels

Export fields
- ☑ Create from attributes
- ☑ Create from single sub-elements
- ☑ Create from text values
- ☑ Convert entities to text

Automatic fields
- ☑ Create primary/foreign keys
- ☑ Independent primary key counter for every element

	Element name	Record count	Field count
☑	Address	1	8
☑	Company	1	5
☑	Person	3	9

4. Click the **Export to Database** button.



Export to database

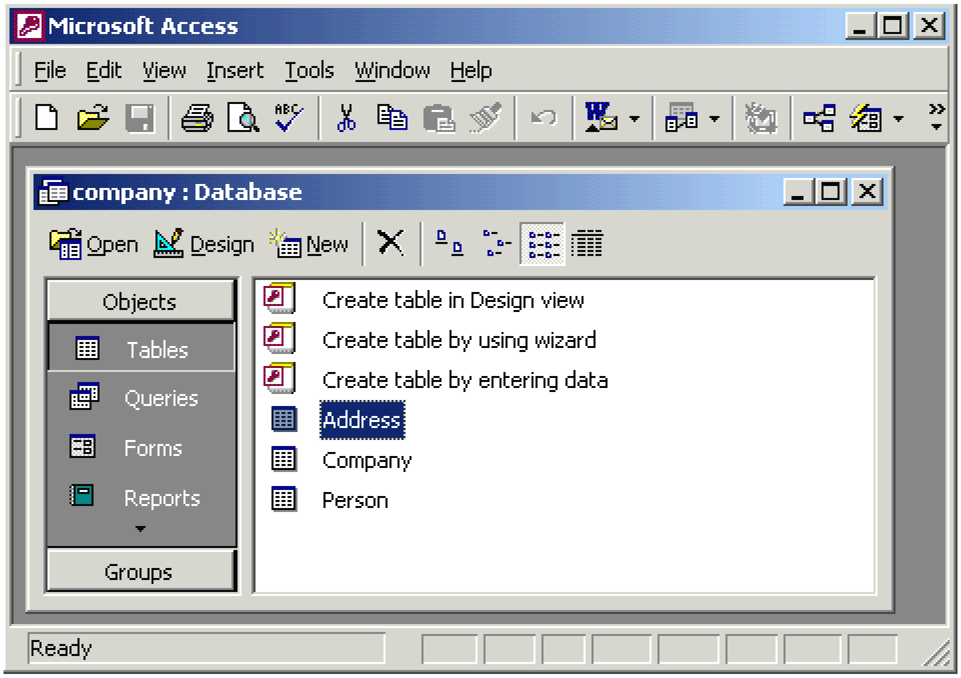
Destination database
- ● Create a new Microsoft Access database
- ○ Choose an existing Microsoft Access database
- ○ Export to following database (enter ADO connection string):
- ○ Export to following database (enter ODBC connection string):

Namespace Options
- ● Exclude Namespace
- ○ Replace colon with underscore

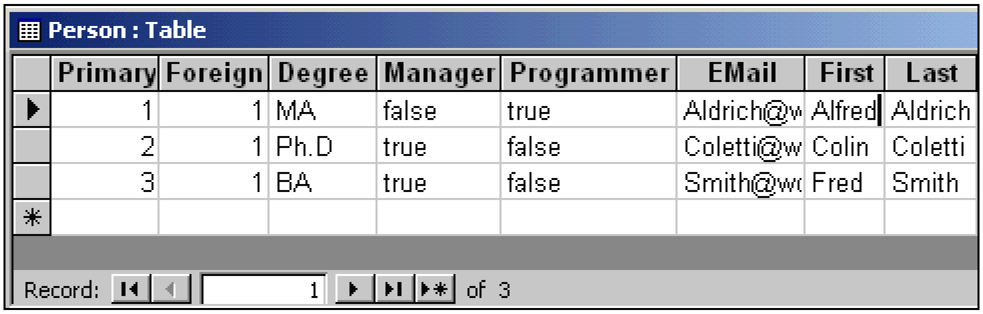
Table structure
- ☑ Create missing tables in the database
- Create text fields with length: 255

This dialog allows you to select if you want to create a new Access table, export data to an existing one, or export to a third party database. The namespace option, "Exclude Namespace" is active by default.

5. Click the **Create a new Microsoft Access database** entry, and confirm with **OK**.
6. Enter the name of the new database in the Save as... dialog (e.g. `Company.mdb`), and confirm with **Save**.
   A progress indicator displays export progress and a message box appears when the process has been completed successfully. Confirm with **OK**.
7. Open the `Company.mdb` file you just saved. The export process automatically created a table for each exported element.


Microsoft Access
File Edit View Insert Tools Window Help
company : Database
Open Design New
Objects
Tables
Queries
Forms
Reports
Groups
Create table in Design view
Create table by using wizard
Create table by entering data
Address
Company
Person
Ready

8. Double-click the **Person** icon to open the Person table.
The table shows all the Person data from the XML file and includes the "Automatic fields" PrimaryKey and ForeignKey, which can be used to index the database data.



Primary	Foreign	Degree	Manager	Programmer	EMail	First	Last
1	1	MA	false	true	Aldrich@w	Alfred	Aldrich
2	1	Ph.D	true	false	Coletti@w	Colin	Coletti
3	1	BA	true	false	Smith@wo	Fred	Smith

Please note:

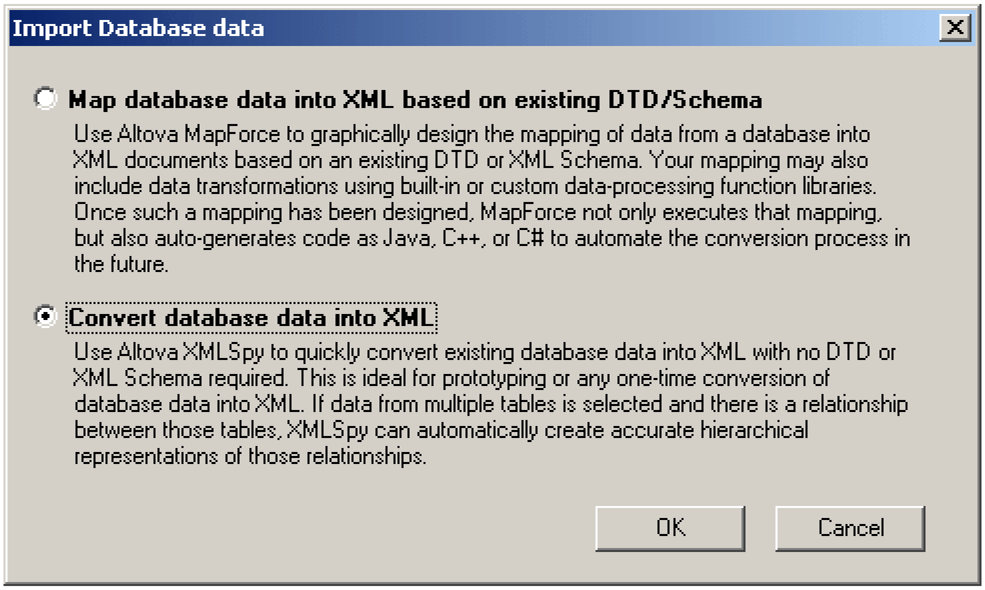
If you select the "Create a new Microsoft Access database" option when **exporting** database data, XMLSpy creates a new Access 2000 database!

If you want to export data to an Access 97 database, please create an empty Access 97 database first, and then select the export option "Choose an existing Access database". There are no restrictions when **importing** data from any Access database.

Importing database data

To import data into XMLSpy:

1. Select the menu option **Convert | Import Database data...**.
The Import database data dialog opens.



Import Database data

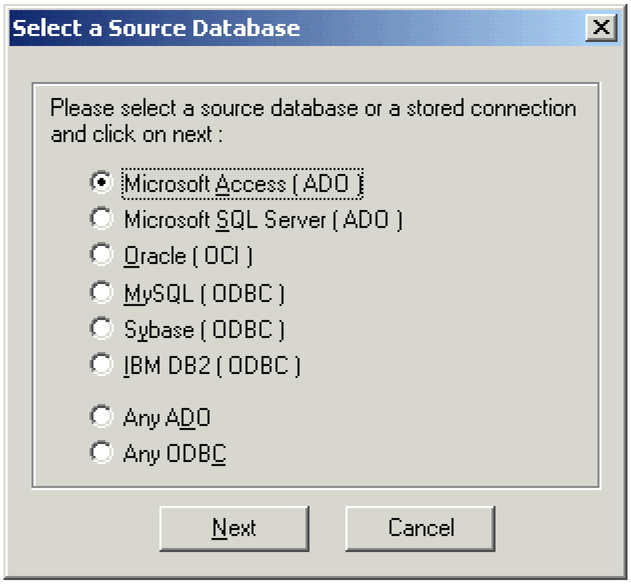
○ Map database data into XML based on existing DTD/Schema

Use Altova MapForce to graphically design the mapping of data from a database into XML documents based on an existing DTD or XML Schema. Your mapping may also include data transformations using built-in or custom data-processing function libraries. Once such a mapping has been designed, MapForce not only executes that mapping, but also auto-generates code as Java, C++, or C# to automate the conversion process in the future.

● Convert database data into XML

Use Altova XMLSpy to quickly convert existing database data into XML with no DTD or XML Schema required. This is ideal for prototyping or any one-time conversion of database data into XML. If data from multiple tables is selected and there is a relationship between those tables, XMLSpy can automatically create accurate hierarchical representations of those relationships.

2. Click the **Convert database data to XML** radio button, and confirm with **OK**. The Select a Source Database dialog opens.

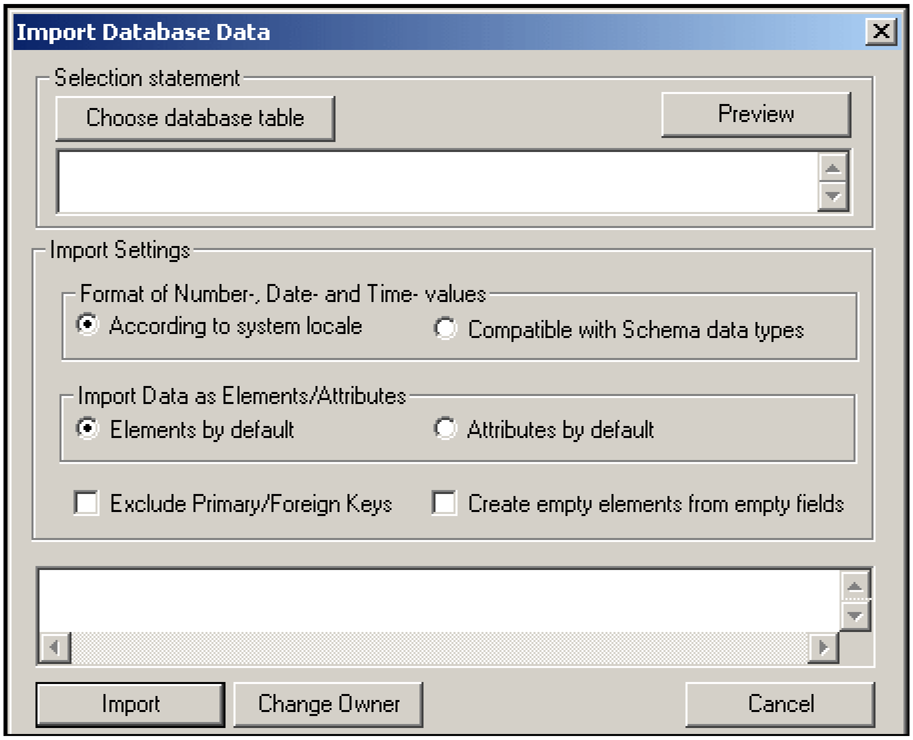


Select a Source Database

Please select a source database or a stored connection and click on next :

● Microsoft Access (ADO)
○ Microsoft SQL Server (ADO)
○ Oracle (OCI)
○ MySQL (ODBC)
○ Sybase (ODBC)
○ IBM DB2 (ODBC)
○ Any ADO
○ Any ODBC

2. Select Microsoft Access (ADO), and click **Next**. The Select a MS Access Database dialog opens.
3. In the Select a MS Access Database dialog, click the **Browse** button, and browse for the **Company.mdb** file. Then click **Next**. This opens the "Import Database Data" dialog box.



Import Database Data

Selection statement

Choose database table Preview

Import Settings

Format of Number-, Date- and Time- values

● According to system locale ○ Compatible with Schema data types

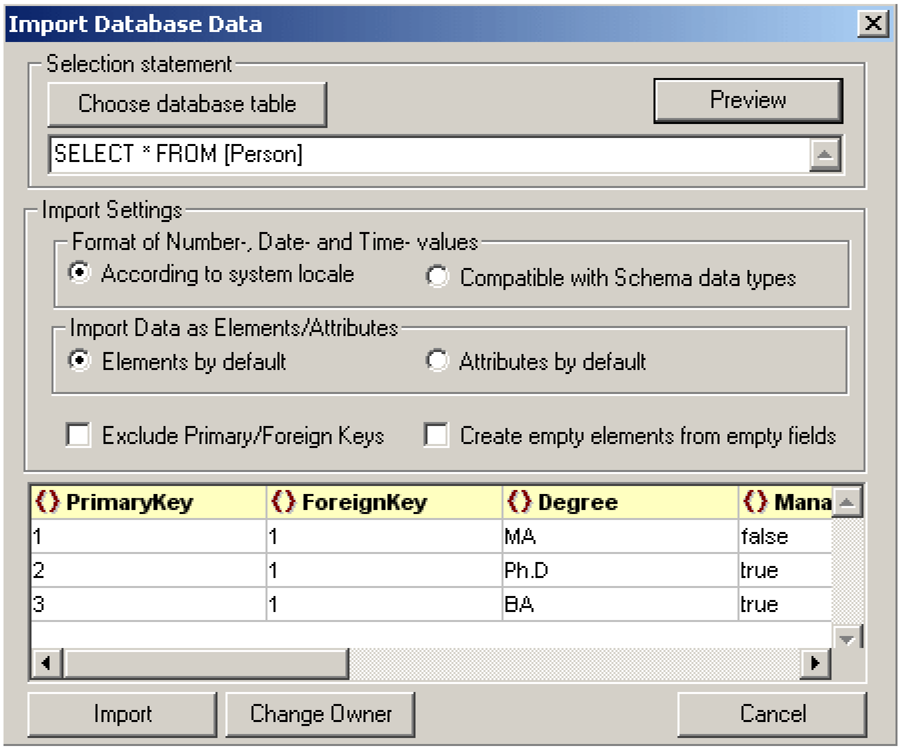
Import Data as Elements/Attributes

● Elements by default ○ Attributes by default

□ Exclude Primary/Foreign Keys □ Create empty elements from empty fields

Import Change Owner Cancel

4. Click the **Choose database table** button, select Person, and confirm with **OK**. This inserts a Select statement in the text box: `SELECT * FROM [Person]`.
5. Click the **Preview** button to see a preview of the selection in the lower pane of the dialog. The Preview window displays only those records that fulfill the select criteria.



Import Database Data

Selection statement

Choose database table Preview

SELECT * FROM [Person]

Import Settings

Format of Number-, Date- and Time- values

● According to system locale ○ Compatible with Schema data types

Import Data as Elements/Attributes

● Elements by default ○ Attributes by default

□ Exclude Primary/Foreign Keys □ Create empty elements from empty fields

{} PrimaryKey	{} ForeignKey	{} Degree	{} Mana
1	1	MA	false
2	1	Ph.D	true
3	1	BA	true

Import Change Owner Cancel

Importing fields as: attributes, elements, or skip import

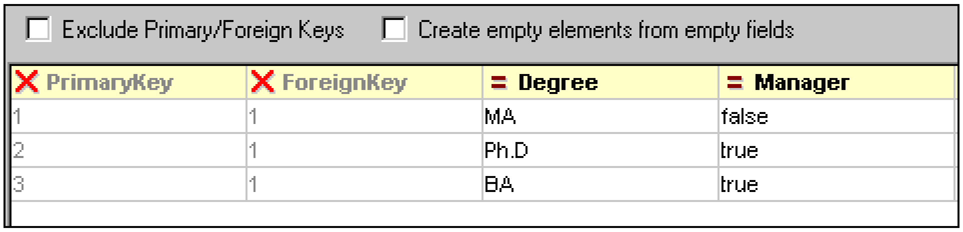
By default, all fields are set to be imported as elements. The preview window allows you to specify the XML node type (element or attribute) for each field or whether a field should not be imported. **Clicking** repeatedly on the **element symbol** {} to the left of the element name, cycles through the available possibilities:

{}	Import this field as an **Element**.
=	Import this field as an **Attribute**.
☒	**Skip**; do not import this field.

Now specify the XML node type for each field, or, alternatively, that the field should not be imported.

Do the following:

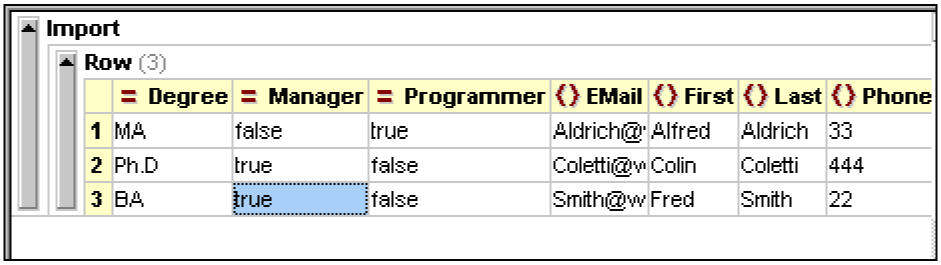
1. Click the **Element** symbol of the PrimaryKey column, until the **skip** symbol appears. Do the same for the ForeignKey column.
2. Click the **Element** symbol of the Degree column, until the **attribute** symbol appears. Do the same for the Manager and Programmer columns.



□ Exclude Primary/Foreign Keys □ Create empty elements from empty fields

✕ PrimaryKey	✕ ForeignKey	= Degree	= Manager
1	1	MA	false
2	1	Ph.D	true
3	1	BA	true

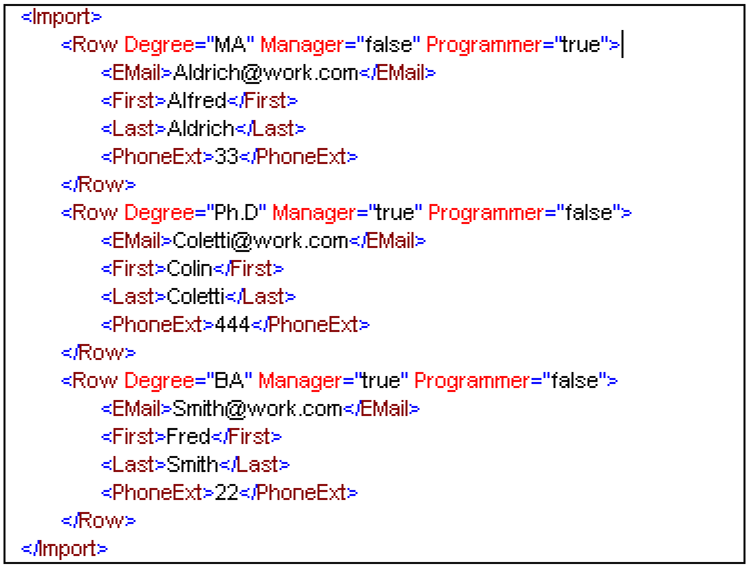
3. Click the **Import** button to start the import process. XMLSpy creates an **untitled XML file** containing the Person table data. The **document element** is called Import, and each Person table has been imported as a Row element.



Import
Row (3)

	= Degree	= Manager	= Programmer	{} EMail	{} First	{} Last	{} Phone
1	MA	false	true	Aldrich@	Alfred	Aldrich	33
2	Ph.D	true	false	Colletti@w	Colin	Coletti	444
3	BA	true	false	Smith@w	Fred	Smith	22

Click the **Text** tab to get another view of the imported data.



```
<Import>
    <Row Degree="MA" Manager="false" Programmer="true">
        <EMail>Aldrich@work.com</EMail>
        <First>Alfred</First>
        <Last>Aldrich</Last>
        <PhoneExt>33</PhoneExt>
    </Row>
    <Row Degree="Ph.D" Manager="true" Programmer="false">
        <EMail>Coletti@work.com</EMail>
        <First>Colin</First>
        <Last>Coletti</Last>
        <PhoneExt>444</PhoneExt>
    </Row>
    <Row Degree="BA" Manager="true" Programmer="false">
        <EMail>Smith@work.com</EMail>
        <First>Fred</First>
        <Last>Smith</Last>
        <PhoneExt>22</PhoneExt>
    </Row>
</Import>
```

For more information on importing data, see Import DB data in the User Reference section of the XMLSpy User Manual and Programmer's Reference.

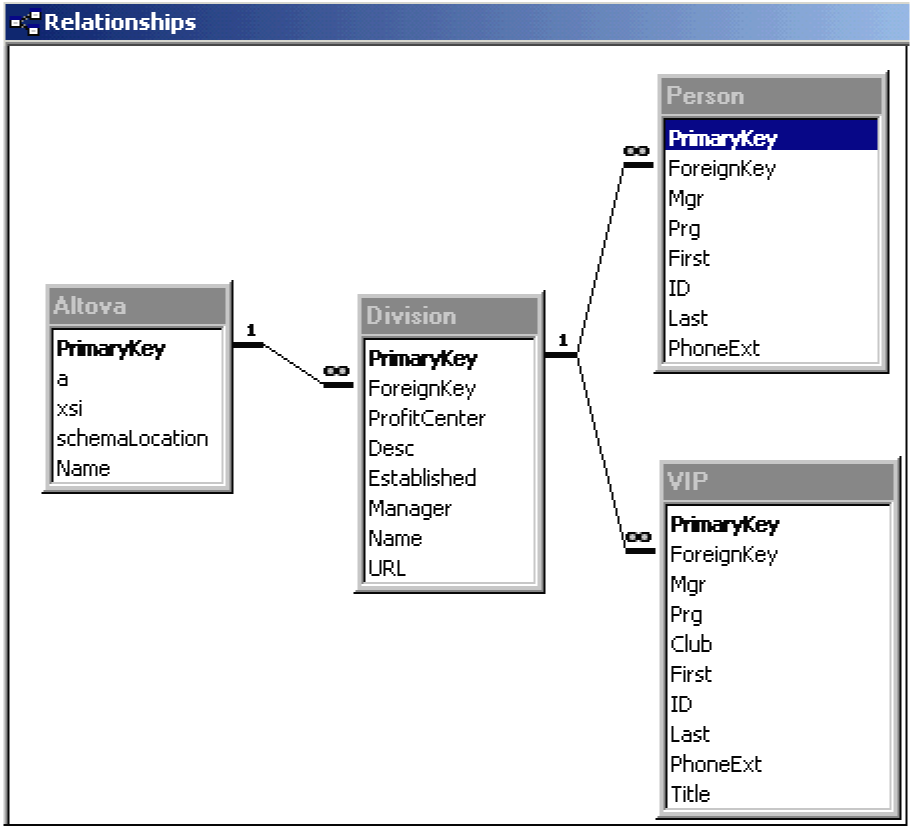
Creating a database schema

XMLSpy enables you to create a schema based upon an external database file. Most industry standards, as well as ADO and ODBC compatible databases, are supported.

Goal of this section:
To convert an existing MS Access database into a schema file, having the same table structure.

This will be achieved by:

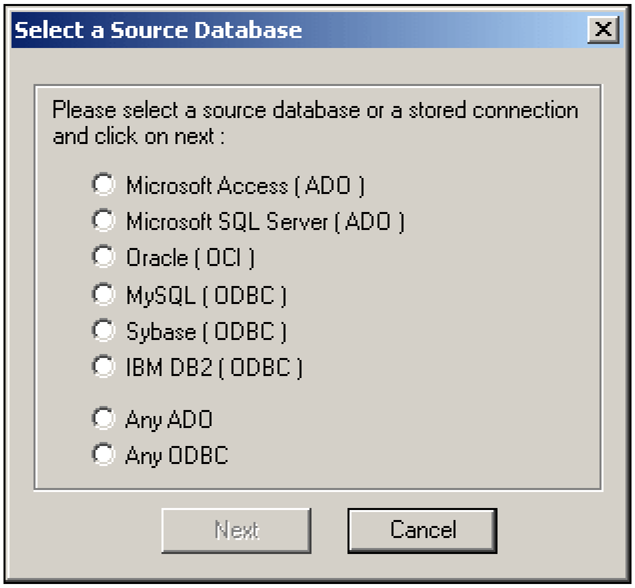
- Using the menu option **Convert | Create XML Schema from DB Structure** to create the schema in XMLSpy.
- This example uses the **DB2Schema.mdb** file supplied with this tutorial. The Relationships view of the DB2schema.mdb file is visible in the diagram below. Use the menu option **Tools | Relationships** in MS Access to view the relationships.



Converting a database to an XML schema

To create a schema from a database file:

1. In XMLSpy, select the menu option **Convert | Create XML Schema from DB Structure.** The Select a Source Database dialog opens.



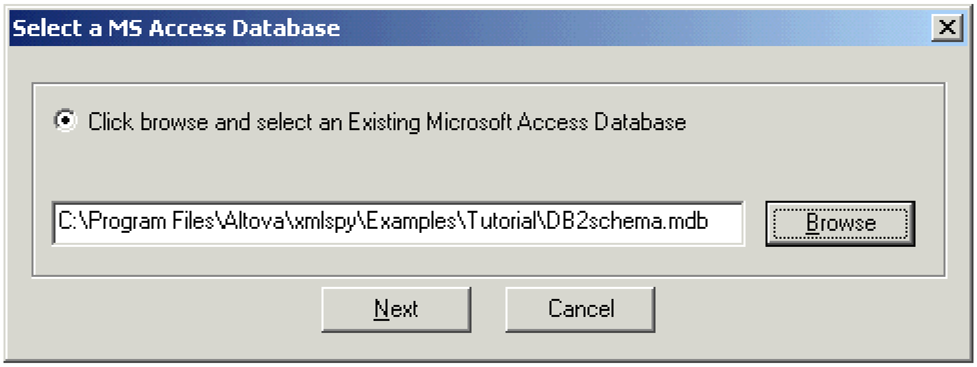
Please select a source database or a stored connection and click on next :

○ Microsoft Access (ADO)
○ Microsoft SQL Server (ADO)
○ Oracle (OCI)
○ MySQL (ODBC)
○ Sybase (ODBC)
○ IBM DB2 (ODBC)

○ Any ADO
○ Any ODBC

2. Select **Microsoft Access (ADO)**, and click the **Next** button.

3. Click the **Browse** button and select the file **DB2schema.mdb** in the `Examples\Tutorial` folder. This enters the **DB2schema.mdb** filename in the dialog.

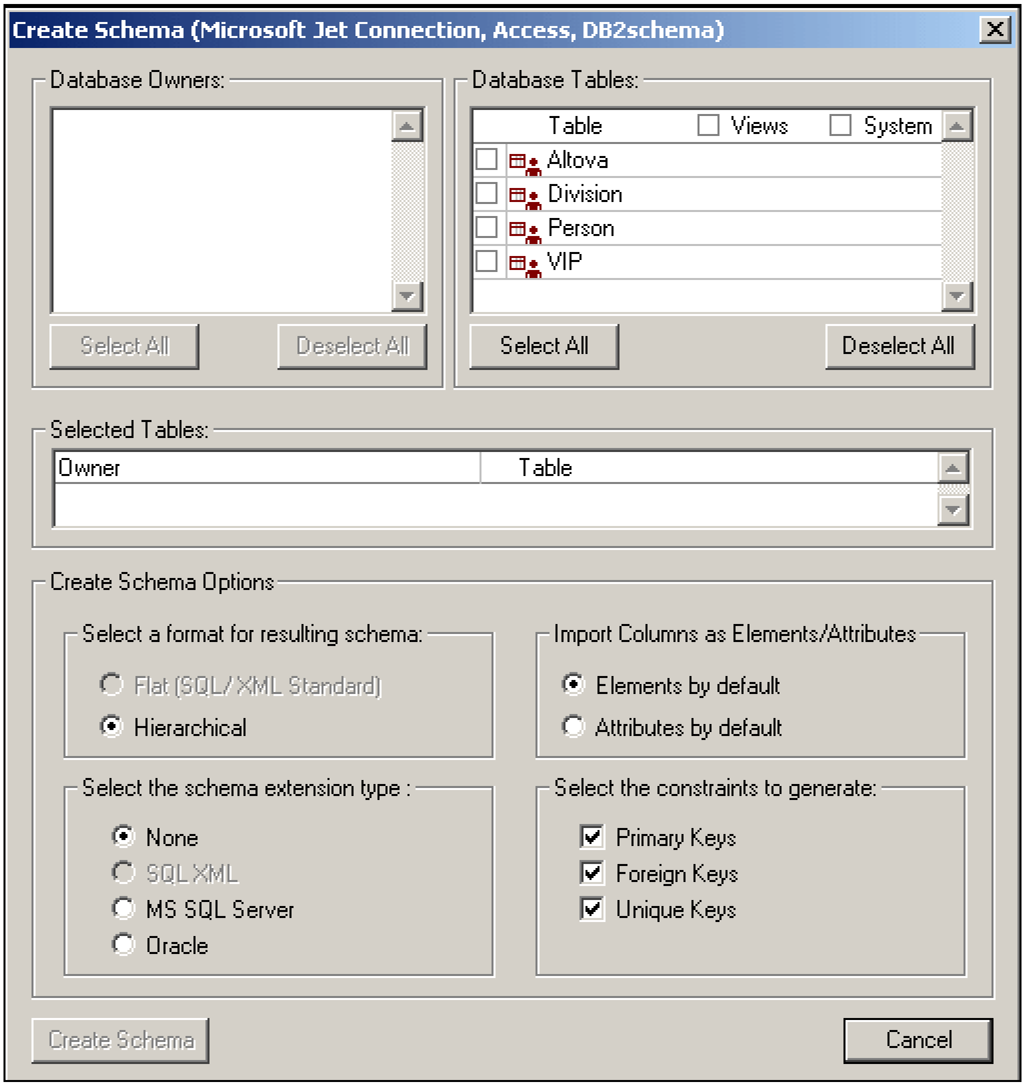


Select a MS Access Database

● Click browse and select an Existing Microsoft Access Database

C:\Program Files\Altova\xmlspy\Examples\Tutorial\DB2schema.mdb

4. Click the **Next** button to continue. This opens the **Create Schema** dialog, which allows you to define specific database tables and other schema settings.



Create Schema (Microsoft Jet Connection, Access, DB2schema)

Database Owners:

Database Tables:

Table	□ Views	□ System
□ Altova		
□ Division		
□ Person		
□ VIP		

Selected Tables:

Owner	Table

Create Schema Options

Select a format for resulting schema:
○ Flat (SQL/XML Standard)
● Hierarchical

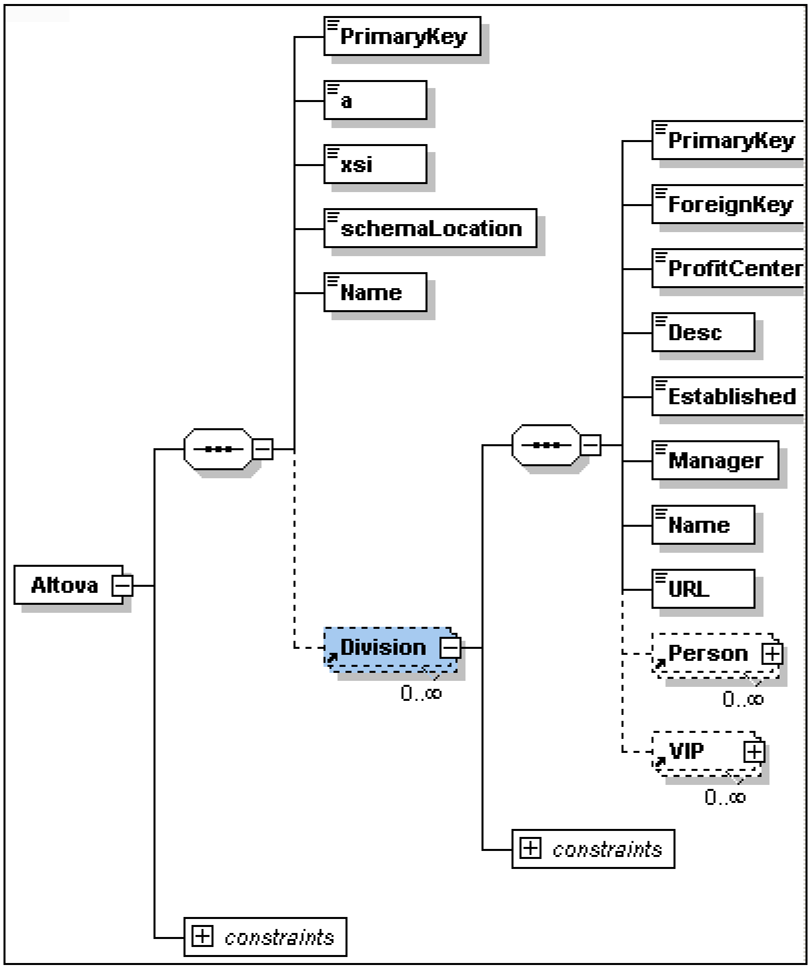
Import Columns as Elements/Attributes
● Elements by default
○ Attributes by default

Select the schema extension type :
● None
○ SQL XML
○ MS SQL Server
○ Oracle

Select the constraints to generate:
☑ Primary Keys
☑ Foreign Keys
☑ Unique Keys

5. In the Database Tables pane, click the **Select All** button to select all the data tables. The selected tables are displayed in the Selected Tables pane.

6. Click the **Create Schema** button to start the conversion process. The generated schema appears in the Schema/WSDL Design View.
7. Click the "Identity constraints" tab, to see the keyref and key fields of the respective elements.
8. Click the component icon next to the **Altova** global element, to see the content model.



9. Select the menu option **File | Save as**, and save the new schema e.g. **DB2schema.xsd**.
10. Click the Display all globals icon , to return to the schema overview.

Databases currently supporting the key and keyref fields:

MS Access and several other databases are able to automatically provide the key and keyref information for the ADO driver that is used to create the database hierarchy.

Other DBs: converting DB structure to XML Schema

For databases other than MS Access, the conversion is a two-step process:

- Building the connection string to connect to the DB
- Selecting the DB tables to convert

For details about converting from these DBs, see Creating XML Schema from DB Structure in the User Reference section of the XMLSpy User Manual and Programmer's Reference.

2.1.8 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.

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.

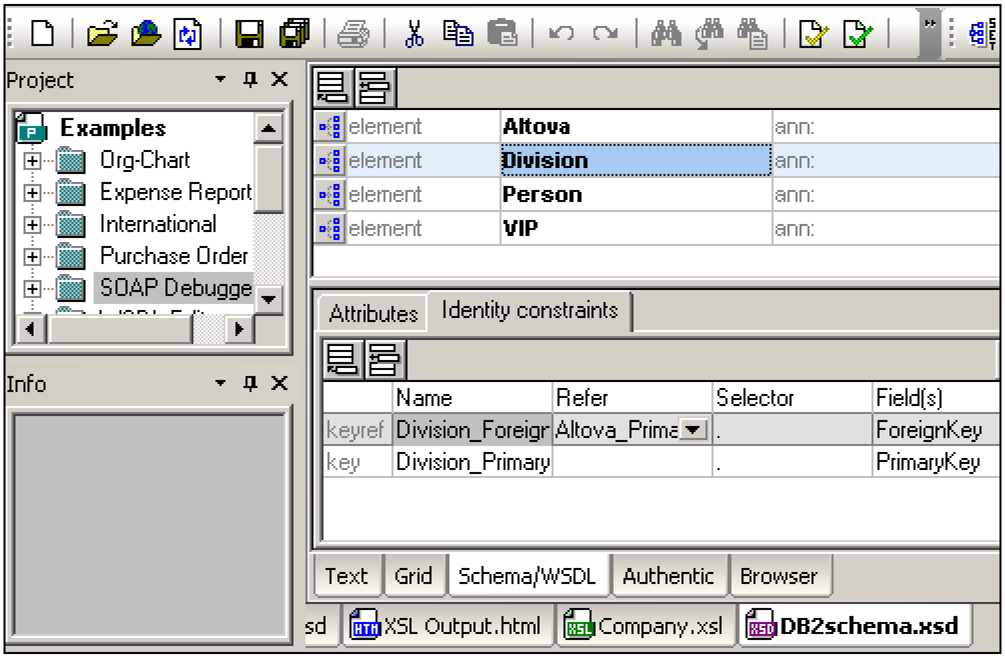
All the above project settings can be defined using the menu option **Project | Project Properties...**. In the next section, you will create a project using the Project menu.

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.

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.



element	Altova	ann:
element	Division	ann:
element	Person	ann:
element	VIP	ann:

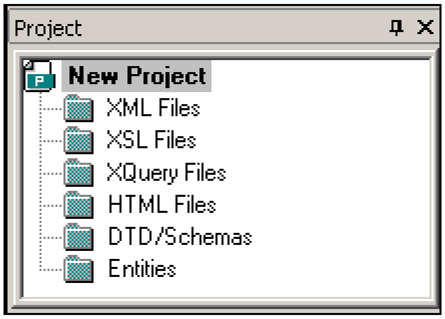
	Name	Refer	Selector	Field(s)
keyref	Division_Foreigr	Altova_Prima	.	ForeignKey
key	Division_Primary		.	PrimaryKey

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 `DB2schema.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.

2.1.9 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.

2.2 Authentic View Tutorial

In Authentic View, you will open an existing XML file that is linked to a StyleVision Power Stylesheet. You then modify the file using the various Authentic View features. The tutorial consists of three main parts:

- Opening an existing XML file in Authentic View
- Editing data (adding new document components as well as content); this section forms the bulk of the tutorial
- Printing out 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 sections.

Tutorial requirements
All **the files** you need for the tutorial are in the `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 in the case of the free 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.

2.2.1 Opening an XML document in Authentic View

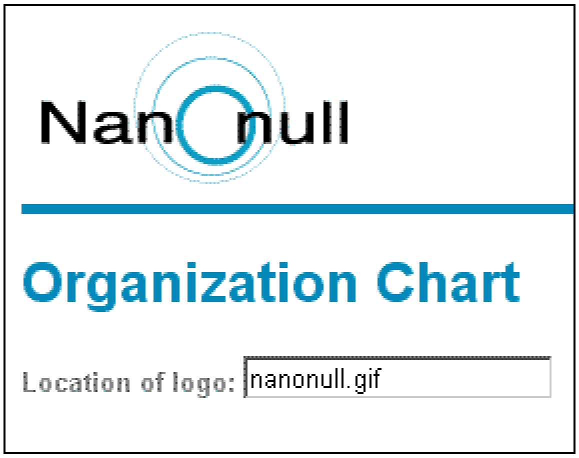
The file `NanonullOrg.xml` is in the `Examples` folder in the folder where you have installed the application. Typically, the path to the file would be:
`c:\Program Files\Altova\ApplicationName\Examples\NanonullOrg.xml`

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.

The file `NanonullOrg.xml` opens directly in Authentic View. This is because:

1. The file already has a StyleVision Power Stylesheet assigned to it.
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.)



Organization Chart

Location of logo: nanonull.gif

Remember: It is the StyleVision Power Stylesheet that defines and controls how an XML document is displayed in Authentic View. Without a StyleVision Power Stylesheet, there can be no Authentic View of the document.

Please note: Alternatively, you could open an XML template in Authentic View by selecting a StyleVision Power Stylesheet.

To open such a template:

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 StyleVision Power Stylesheet.

If a Template XML File has been assigned to the StyleVision Power Stylesheet, then the data in the Template XML File is used as the starting data of the template that has been created in Authentic View.

2.2.2 Entering data in Authentic View

Entering data in Authentic View can be as simple as the associated StyleVision Power Stylesheet makes it.

In the **simplest user scenario**, you will enter content as free-flowing text or into data-input fields, or you will make a selection from a list of user options. In short, you focus on entering content; user-side structural modification (adding elements, tables, etc) and document formatting is kept to a minimum. These restrictions help ensure the validity of the document and the accuracy of data. They also keep you focused on the content.

In most cases, however, you will be given the **option of adding a few elements**. These additions can be implicit, as when you press Enter to add a new paragraph element or click an icon to mark text bold. Or they can be explicit, as when you append an element via the Elements Entry Helper. In the latter kind of scenario, you would require a working knowledge of how the document is structured.

At the other extreme, users may be given complete freedom to structure the document. To do this, however, you would require a good knowledge of the schema on which the document is based.

In Authentic View, you can enter or edit the following types of data and data structures:

- Element content (can be entered as a text entry or via a data-entry device)
- Attribute values (can be entered as a text entry or via a data-entry device, or as a value in the Attributes Entry Helper)
- Entities (can be inserted via the Entities Entry Helper)
- Elements (can be added, changed to other elements, or deleted)
- XML tables (can be inserted, and their structure, formatting, and content specified)

This tutorial shows you how to manipulate elements and enter content in Authentic View. XML tables are discussed in detail in the Using tables in Authentic View section.

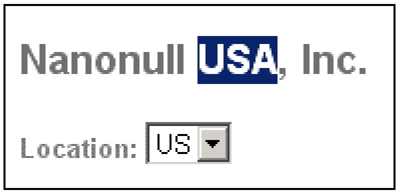
2.2.3 Adding document content and elements

Adding text content

You can enter element content and attribute values directly as text. To insert content, place the cursor at the location where you want to insert the text, and type. You can also edit such content by highlighting the text, and typing in the replacement text or deleting the highlighted text.

To change the name of the company:

- Place the cursor after Nanonull, and type in USA to change the name from Nanonull, Inc. to Nanonull USA, Inc.



Nanonull USA, Inc.

Location: US

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.

Please note: In Hide markup mode, an empty element can easily be overlooked. To make sure that you are not overlooking an empty element, switch to Show large markup or Show small markup mode.

Adding content via a data-entry device

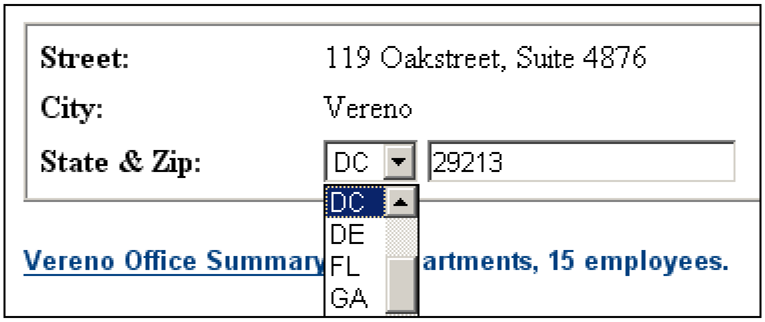
In the content editing you have learned above, content is added by directly typing in text as element content. There is one other way that **element content** (or attribute values) can be entered in Authentic View: via data-entry devices.

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.



Street: 119 Oakstreet, Suite 4876

City: Vereno

State & Zip: DC 29213

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>
```

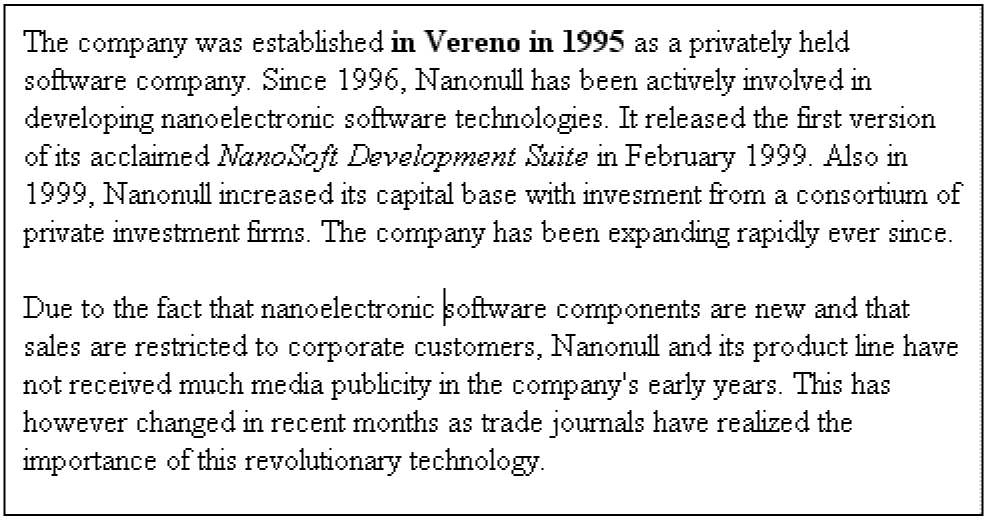
Notice that the combo-box selection "DC" is mapped to a value of "DC". The value of the Zip field is entered directly as content of the `ipo:zip` element.

Adding elements = inserting before or appending
You can add an element by **inserting it before** or **appending it** to the **current element** (the element in which the cursor is).

Add another paragraph to the description of the company. This involves appending the paragraph element (`para`, in this case) and entering content for the element.

Do the following:

1. Place the cursor in the last paragraph of the description text.



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.

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.

2. In the Elements Entry Helper, click the Insert Before Element icon that is located to the left of the `para` element. This **inserts** a `para` **before** the current `para`—which is not where you want the new `para` element. Click **Undo**.
3. In the Elements Entry Helper, click the Append Element icon that is located to the left of the `para` element. This **appends** a `para` element, which is what we want.
4. Type in some text as the content of `para` at the blinking cursor.

Alternatively, to append a `para` (or any paragraph-type) element, press **Enter** at the point where you want to append the new element.

Please note: You can also use the Enter key to insert and append list items in numbered lists and itemized (bulleted) lists.

Adding elements in the document

It is important to remember that **only same- or higher-level elements** can be inserted before or appended after the current 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 it as a sibling of the relevant ancestor. For example, if a section element is appended to a paragraph, it is created as a sibling of the section that contains the current paragraph element, and it is created as the last sibling section.

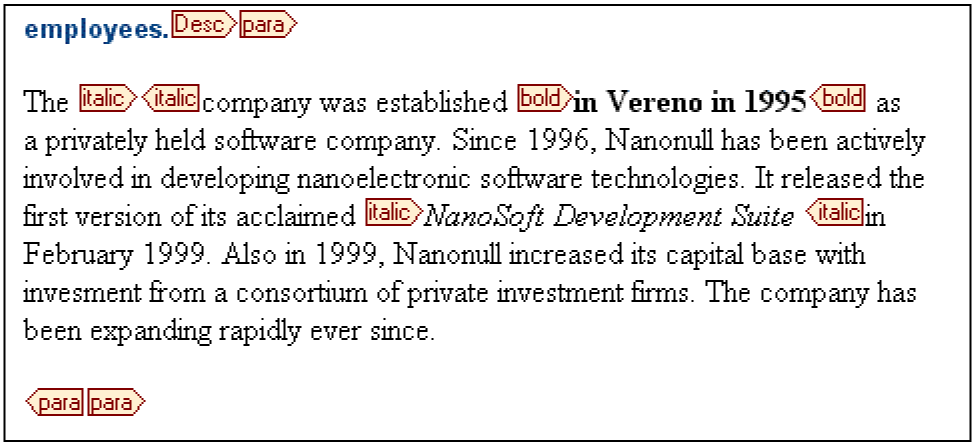
At any given location in the document, the elements you can insert before or append are shown in the Elements Entry Helper with the icons to their left.

2.2.4 Inserting an element

You can **insert** an element **as a child** of the **current element** (the element in which the cursor is).

In `NanonullOrg.xml`, the `para` element can contain the elements `italic` and `bold`. Now experiment with inserting these elements, as follows:

1. Place the cursor at a location in the `para` element where you want to insert the child element (check the status bar at bottom left for your location in the document). Notice that in the Elements Entry Helper, the `italic` and `bold` elements are listed with (Insert Element) icons next to the listing.
2. Click the icon next to the `italic` element. This inserts the `italic` element. Switch to Show large markup mode to see the element tags.



employees. Desc para

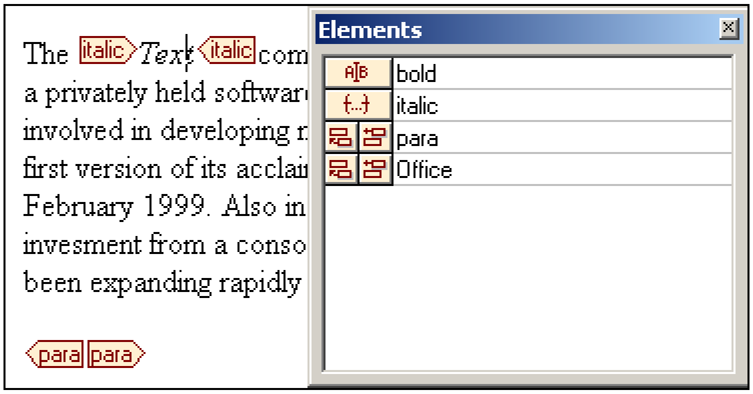
The italic italic company was established bold **in Vereno in 1995** bold as a privately held software company. Since 1996, Nanonull has been actively involved in developing nanoelectronic software technologies. It released the first version of its acclaimed italic *NanoSoft Development Suite* italic 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.

para para

3. Type in the content of the element between the italic element tags.

When the cursor is within an element, any element which can be inserted as a child is shown in the Elements Entry Helper with the Insert Element icon next to it.

Now place the cursor inside the `italic` element, and look at the Elements Entry Helper.



The italic *Text* italic com
a privately held softwar
involved in developing n
first version of its acclai
February 1999. Also in
invesment from a conso
been expanding rapidly

para para

The Elements Entry Helper shows that the `bold` element can be inserted as a child of `italic`. Click the Insert Element icon for the `bold` element, and see what happens.

2.2.5 Clearing elements

Clearing an element

When you clear an element, you remove its markup without modifying its content.

To clear the `italic` element, do the following:

1. Switch to Show large markup mode.
2. Place the cursor at an insertion point within the `italic` element. The `italic` element is shown in the Elements Entry Helper with the icon next to it.

3. Click the {...} icon. In the main window, the `italic` element markup, and the italic formatting, is removed.

You can also select a text fragment within the `italic` element (instead of placing the cursor as an insertion point), and clear the `italic` element. The only difference is that the `italic` element in the Elements Entry Helper will be shown with a second variant of the Clear Element icon next to it: {...}. This icon does the same thing as the {...} icon, but only for the selected text.

Please note: It is important to place the cursor within the element content and not to select the entire element. Selecting the entire element causes the Apply Element context to be activated instead, which allows you to replace the selected element with another element.

2.2.6 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 free-flowing 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 Adding document content and elements.

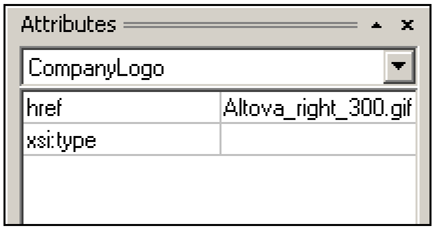
In such cases, the distinction between element content and attribute value is made by the StyleVision Power Stylesheet and the data is handled appropriately.

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.

The location of the logo that is used in `NanonullOrg.xml` 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).



CompanyLogo

href	Altova_right_300.gif
xsi:type	

This causes the Nanonull logo to be replaced by the Altova logo.

Please note: If you are required to enter the value of an attribute, the designer of the StyleVision Power Stylesheet will, typically, include an input mechanism for this data in Authentic View.

2.2.7 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.

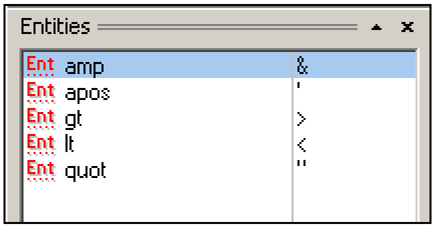
All the entities available to you are displayed in the Entities Entry Helper. To insert an entity, double-click it.

In `NanonullOrg.xml`, change the title of Joe Martin (in Marketing) to Marketing Manager Europe & Asia.

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. In `NanonullOrg.xml`, entities have been declared for these characters, and are therefore displayed in the Entities Entry Helper.

To insert the ampersand entity in the title, "Marketing Manager Europe & Asia":

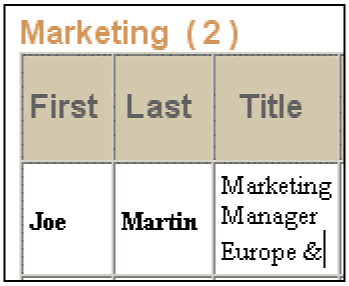
1. Place the cursor where the ampersand is to be inserted.
2. Double-click the entity listed as "amp".



Entities

Ent amp	&
Ent apos	'
Ent gt	>
Ent lt	<
Ent quot	"

This inserts an ampersand.



Marketing (2)

First	Last	Title
Joe	Martin	Marketing Manager Europe &

Please 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** (**F8**).

Also see: Attributes Entry Helper under Authentic View entry helpers.

2.2.8 Printing the document

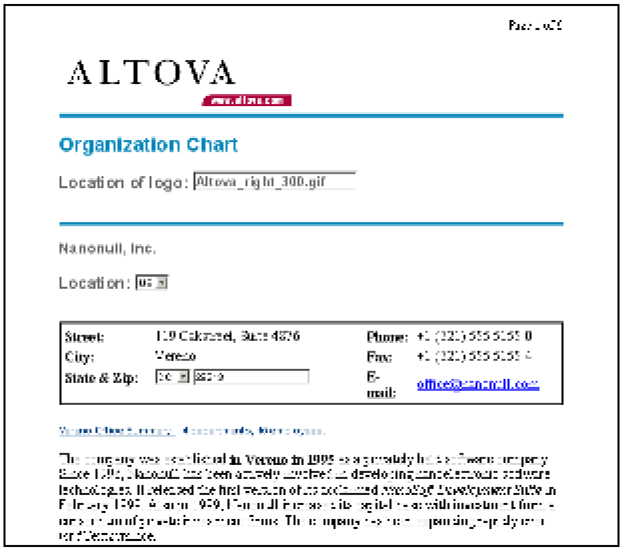
A printout from Authentic View of an XML document preserves the formatting seen in Authentic View.

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%.



[illegible]

ALTOVA

Organization Chart

Location of logo: Altova_right_300.gif

Nanonull, Inc.

Location: US

Street:	119 Oakstreet, Suite 4876	Phone:	+1 (321) 555 5155 0
City:	Vereno	Fax:	+1 (321) 555 5155 4
State & Zip:	DC [illegible]	E-mail:	office@nanonull.com

Vereno Office Summary: 4 departments, [illegible] employees.

The company was established in Vereno in 1995 as a privately held software company. Since 1996, Nanonull has been actively involved in developing nanoelectronic software technologies. It released the first version of its acclaimed NanoSoft Development Suite in February 1999. Also in 1999, Nanonull increased its capital base with investment from a consortium of private investment firms. The company has been expanding rapidly ever since.

Notice that the formatting of the page is the same as that in Authentic View.

3. To print the file, click **File | Print**.

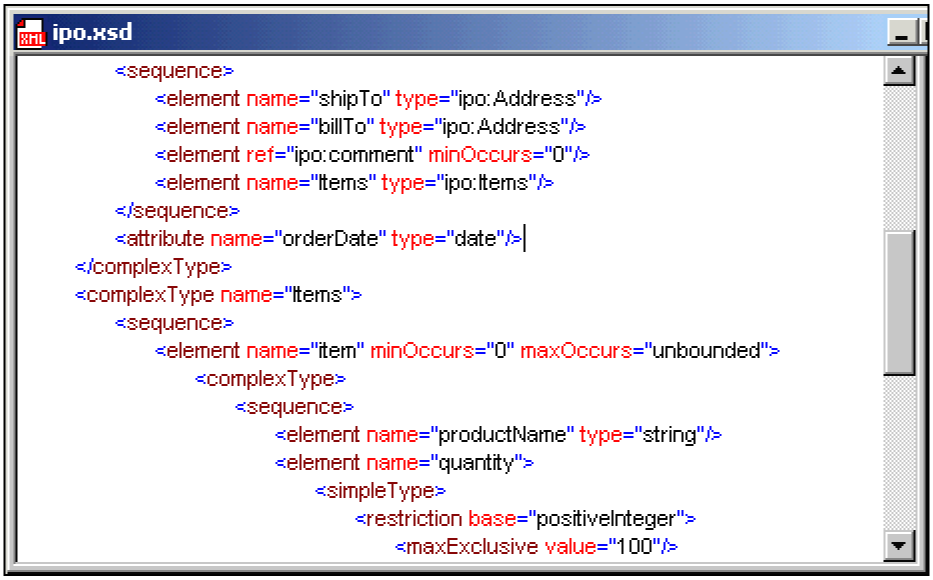
Note that you can also print a version of the document that displays (small) markup. To do this, switch Authentic View to Show small markup mode or Show large markup mode, and then print. Both modes produce a printout that displays small markup.

3 Text View

In Text View, you can type in your document text directly, i.e. markup and content. Text View provides a number of features to help you quickly and accurately type in your document. Among the main features are the following:

- Visual features to help you read the document more easily. These include customizable syntax-coloring (including the ability to highlight server-side VBScript or JScript code in ASP pages), line-numbering, bookmarking, expandable and collapsible elements, indentation, customizable fonts, and text-wrapping.
- Intelligent editing features like auto-completion of tags and automatic entry of attributes and children.
- Context-sensitive entry helpers, which list allowed elements, attributes, and entities at the cursor insertion point, and allow you to insert these into the document.
- Drag-and-drop and copy-and-paste capabilities.

These features are described in more detail in the rest of this section.



```
        <sequence>
            <element name="shipTo" type="ipo:Address"/>
            <element name="billTo" type="ipo:Address"/>
            <element ref="ipo:comment" minOccurs="0"/>
            <element name="Items" type="ipo:Items"/>
        </sequence>
        <attribute name="orderDate" type="date"/>
    </complexType>
    <complexType name="Items">
        <sequence>
            <element name="item" minOccurs="0" maxOccurs="unbounded">
                <complexType>
                    <sequence>
                        <element name="productName" type="string"/>
                        <element name="quantity">
                            <simpleType>
                                <restriction base="positiveInteger">
                                    <maxExclusive value="100"/>
```

To open the Text View of a document, click the Text button at the bottom of the Document Window or select **View | Text view**. Text view can be used to edit any text file, including non-XML documents.

3.1 Visual Editing Guides in Text View

Text View provides functions to facilitate the editing of large sections of text: line numbering, bookmarks, source folding, and indentation guides. Each of these functions can be enabled or disabled by clicking its respective icon in the toolbar.

Enables/disables line numbering across all open documents

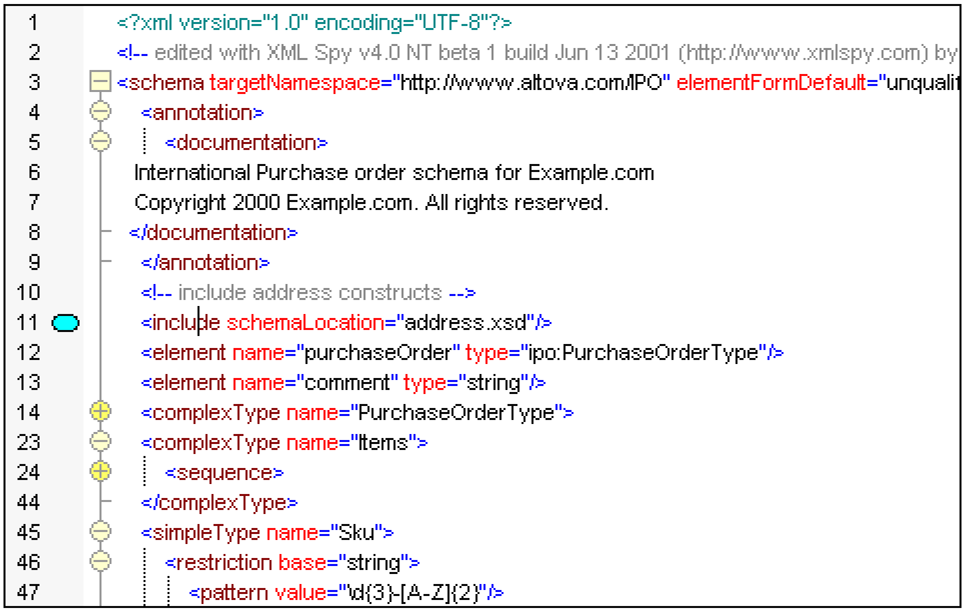
Enables/disables the source folding margin for all open documents

Enables/disables the bookmark margin.

Inserts/removes bookmarks.

Enables/disables indentation guides.

The graphic below shows these features:



```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <!-- edited with XML Spy v4.0 NT beta 1 build Jun 13 2001 (http://www.xmlspy.com) by
3   <schema targetNamespace="http://www.altova.com/IPO" elementFormDefault="unqualif
4     <annotation>
5       <documentation>
6    International Purchase order schema for Example.com
7    Copyright 2000 Example.com. All rights reserved.
8    </documentation>
9     </annotation>
10    <!-- include address constructs -->
11    <include schemaLocation="address.xsd"/>
12    <element name="purchaseOrder" type="ipo:PurchaseOrderType"/>
13    <element name="comment" type="string"/>
14    <complexType name="PurchaseOrderType">
23    <complexType name="Items">
24      <sequence>
44    </complexType>
45    <simpleType name="Sku">
46      <restriction base="string">
47        <pattern value="\d{3}-[A-Z]{2}"/>
```

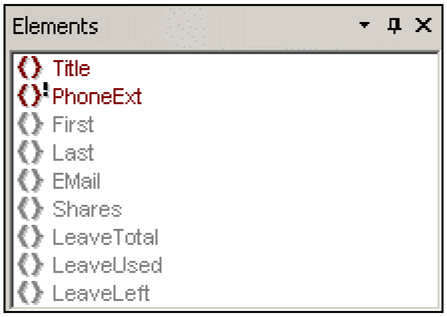
Note the following features in the graphic above:

- Line numbering has been enabled
- Source folding has been enabled. The plus sign indicates a collapsed element; the minus sign indicates an expanded element. When an element is collapsed, the line-numbering is correspondingly collapsed (see the collapsing at Lines 14 and 24). When source folding is disabled, any collapsed element is automatically expanded.
- The Bookmarks Margin has been enabled. A bookmark has been created for Line 11.
- Indentation guides have been enabled. These guides are vertical dotted lines that are helpful in determining where an element starts and ends (see Lines 46 and 47).

3.2 Entry Helpers in Text View

Elements Entry Helper

In Text View, elements that can be entered at the cursor point are displayed in the Elements Entry Helper in dark red. Mandatory elements are listed with an exclamation mark before the element name. Siblings of allowed elements that are themselves not allowed at the cursor point are displayed in gray. When the cursor position changes, the list in the Entry Helper changes to show only those elements that can be inserted at that point (in red) and their siblings (in gray).

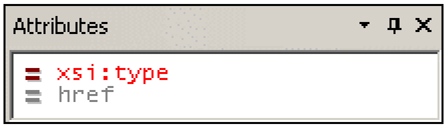


To insert an element at the cursor point, double-click the element you want to insert. The start and end tags of the element are inserted. Mandatory elements are also inserted if this option has been specified in the **Options** dialog (**Tools | Options | Editing**).

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

Attributes Entry Helper

In 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.

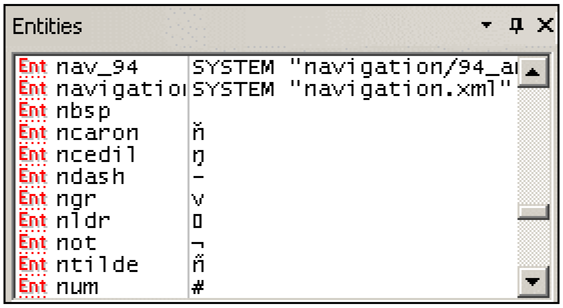


To insert an attribute, double-click the required attribute. The attribute is inserted at the cursor point together with an equals-to sign and quotes to delimit the attribute value. The cursor is placed between the quotes, so you can start typing in the attribute value directly.

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

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.



Ent	nav_94	SYSTEM "navigation/94_a
Ent	navigatio	SYSTEM "navigation.xml"
Ent	nbsp	
Ent	ncaron	ň
Ent	ncedil	ņ
Ent	ndash	–
Ent	ngr	ν
Ent	nldr	□
Ent	not	¬
Ent	ntilde	ñ
Ent	num	#

To insert an entity at the cursor insertion point, double-click the required entity.

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.

3.3 Editing XML Documents

Syntax coloring

Syntax coloring is applied according to XML node kind, that is, depending on whether the XML node is an element, attribute, content, CDATA section, comment, or processing instruction. The text properties of these XML node kinds can be set in the Text Fonts tab of the Options dialog (**Tools | Options**).

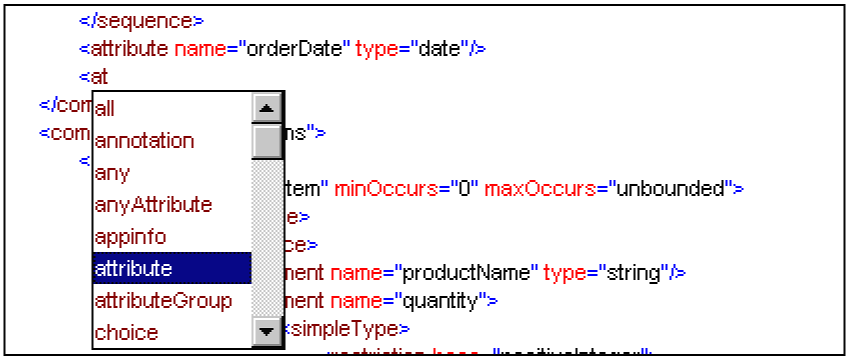
Start-tag and end-tag matching

When you place the cursor inside a start or end tag of an XML element, clicking **Ctrl+E** highlights the other member of the pair. Clicking **Ctrl+E** repeatedly enables you to switch between the start and end tags. This is another 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 DTD or XML 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. Intelligent editing typically works as follows:

1. Type **<** (the less-than character) where you want to insert an XML element. This opens a popup list containing all elements that may be legitimately inserted at that point.
2. Enter the first few characters of the element you want to insert. An element in the popup list containing those characters is highlighted.



```
      </sequence>
      <attribute name="orderDate" type="date"/>
      <at
   </com
   <com                ns">
      <
                       tem" minOccurs="0" maxOccurs="unbounded">
                       e>
                       ce>
                       nent name="productName" type="string"/>
                       nent name="quantity">
                       <simpleType>
```

3. Click on the entry with the mouse pointer or press **Enter** to accept the selected choice. Alternatively, use the arrow keys to highlight your selection and then click or press **Enter**.

The popup window also appears in the following cases:

- If you press the space bar when the cursor is between an element's tags and if an attribute is defined for that element. 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.

Auto-completion

Editing in Text View can easily result in XML documents 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 inserting 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:

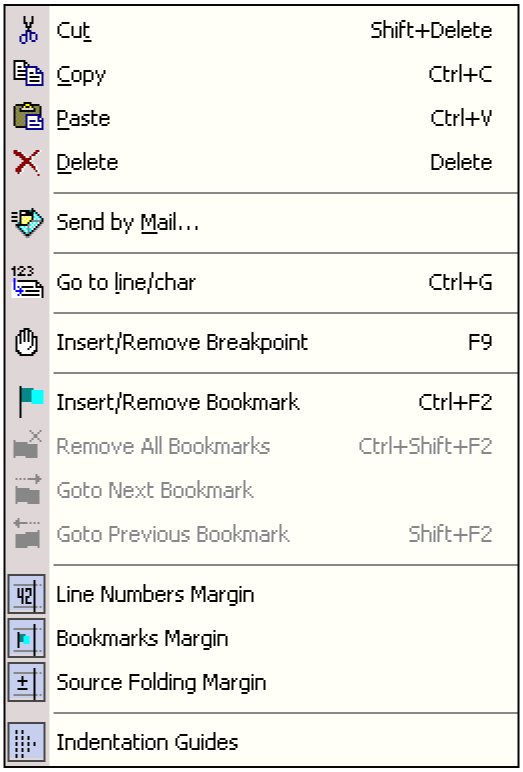
```
<img src="" alt="">|</img>
```

Use the Check well-formedness command at any time to ensure that the document is well-formed. This check is also automatically performed every time you open or save a document.

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.

The other commands in the context menu allow you to manipulate bookmarks and customize Text View.



Find and Replace

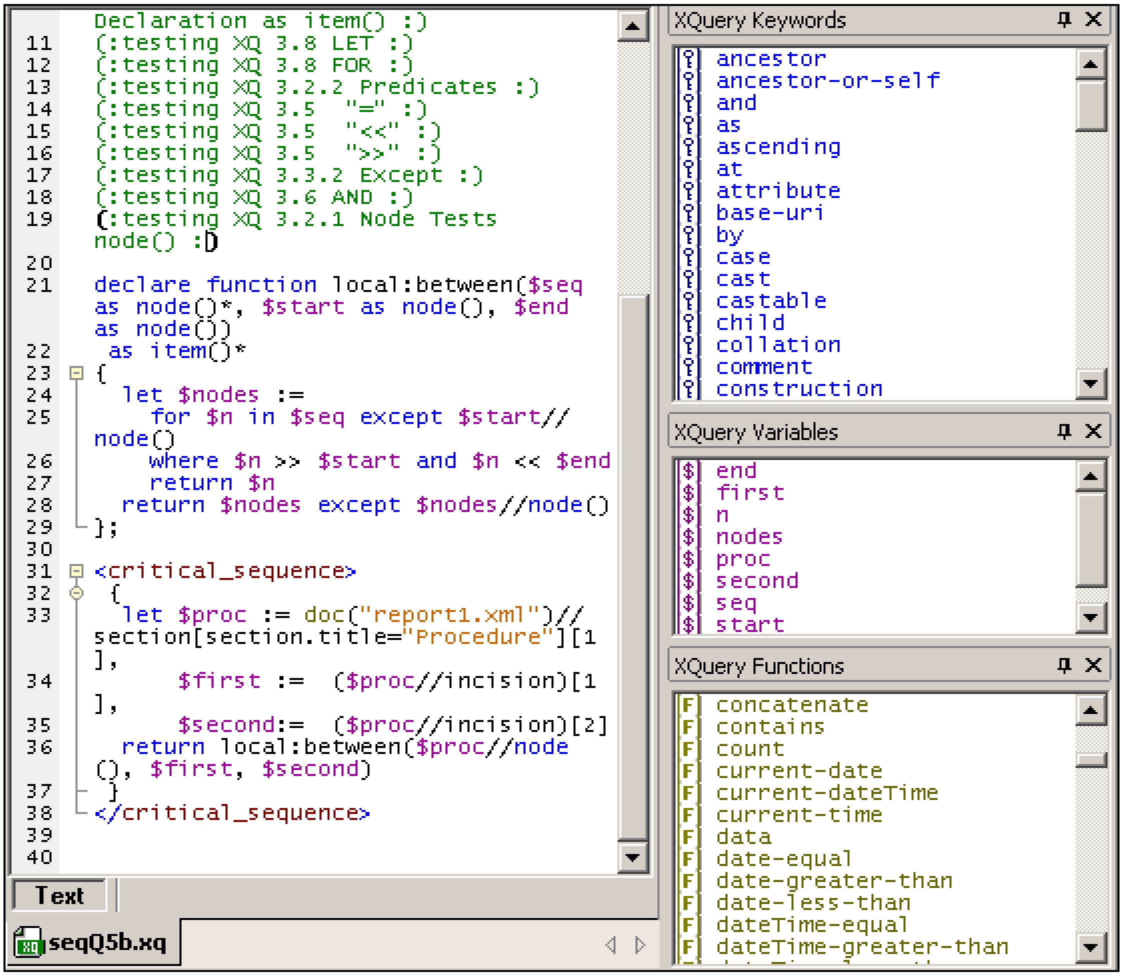
You can use the Find and Replace commands to quickly locate and change text. These commands also take regular expressions as input, thereby giving you powerful search capabilities. (See Edit | Find for details.)

Unlimited Undo

XMLSpy offers unlimited levels of Undo and Redo for all editing operations.

3.4 Editing XQuery Documents

In Text View, you can edit XQuery documents. 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.



```
Declaration as item() :)
(:testing XQ 3.8 LET :)
(:testing XQ 3.8 FOR :)
(:testing XQ 3.2.2 Predicates :)
(:testing XQ 3.5  "=" :)
(:testing XQ 3.5  "<<" :)
(:testing XQ 3.5  ">>" :)
(:testing XQ 3.3.2 Except :)
(:testing XQ 3.6 AND :)
(:testing XQ 3.2.1 Node Tests
node() :)

declare function local:between($seq
as node()*, $start as node(), $end
as node())
 as item()*
{
  let $nodes :=
    for $n in $seq except $start//
node()
    where $n >> $start and $n << $end
    return $n
  return $nodes except $nodes//node()
};

<critical_sequence>
 {
  let $proc := doc("report1.xml")//
section[section.title="Procedure"][1
],
      $first :=  ($proc//incision)[1
],
      $second:=  ($proc//incision)[2]
  return local:between($proc//node
(), $first, $second)
 }
</critical_sequence>
```

Please note: XQuery files can be edited only in Text View. No other views of XQuery files are available.

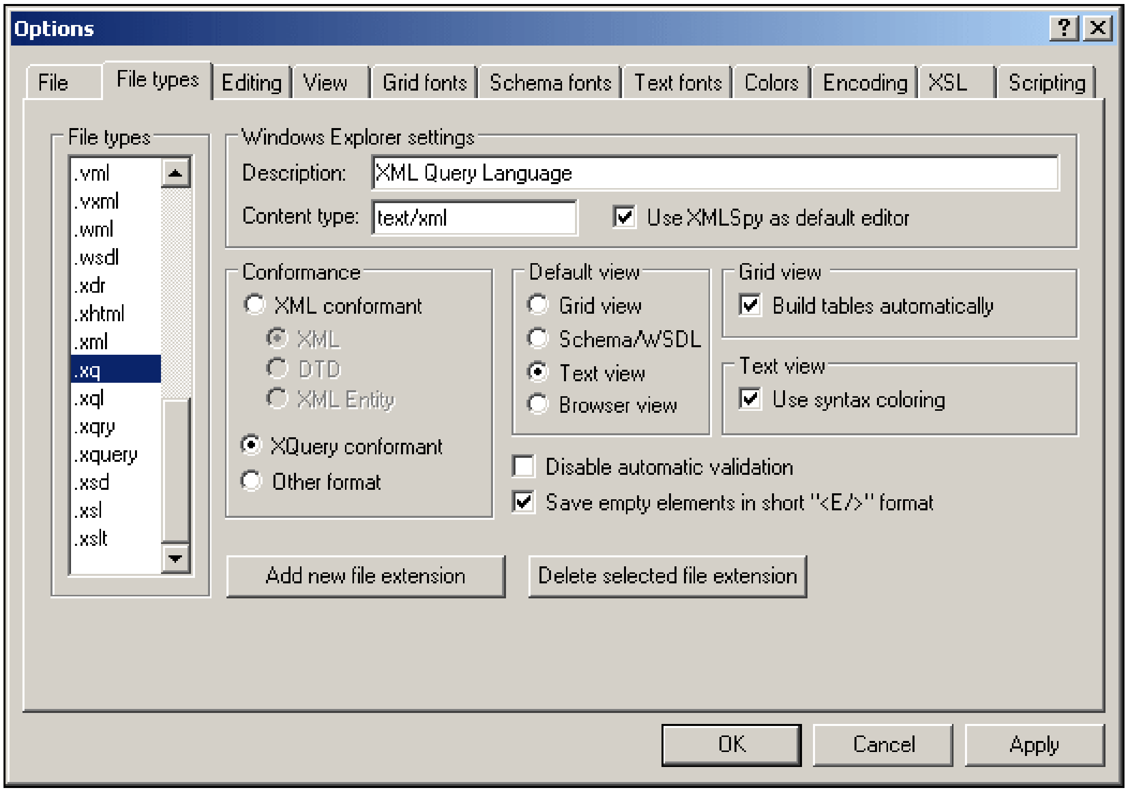
For details about how the Altova XQuery Engine is implemented and will process XQuery files, see XQuery Engine Implementation.

3.4.1 Opening an XQuery Document

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**.



File types: .vml, .vxml, .wml, .wsdl, .xdr, .xhtml, .xml, .xq, .xql, .xqry, .xquery, .xsd, .xsl, .xslt

Windows Explorer settings
Description: XML Query Language
Content type: text/xml
☑ Use XMLSpy as default editor

Conformance
○ XML conformant
● XML
○ DTD
○ XML Entity
● XQuery conformant
○ Other format

Default view
○ Grid view
○ Schema/WSDL
● Text view
○ Browser view

Grid view
☑ Build tables automatically

Text view
☑ Use syntax coloring

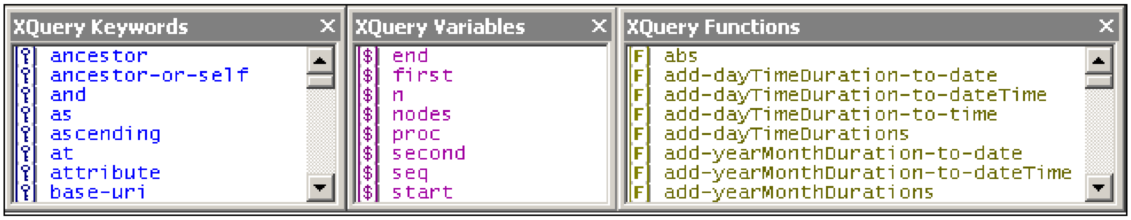
□ Disable automatic validation
☑ Save empty elements in short "<E/>" format

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.

3.4.2 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).



XQuery Keywords: ancestor, ancestor-or-self, and, as, ascending, at, attribute, base-uri

XQuery Variables: end, first, n, nodes, proc, second, seq, start

XQuery Functions: abs, add-dayTimeDuration-to-date, add-dayTimeDuration-to-dateTime, add-dayTimeDuration-to-time, add-dayTimeDurations, add-yearMonthDuration-to-date, add-yearMonthDuration-to-dateTime, add-yearMonthDurations

Note the following points:

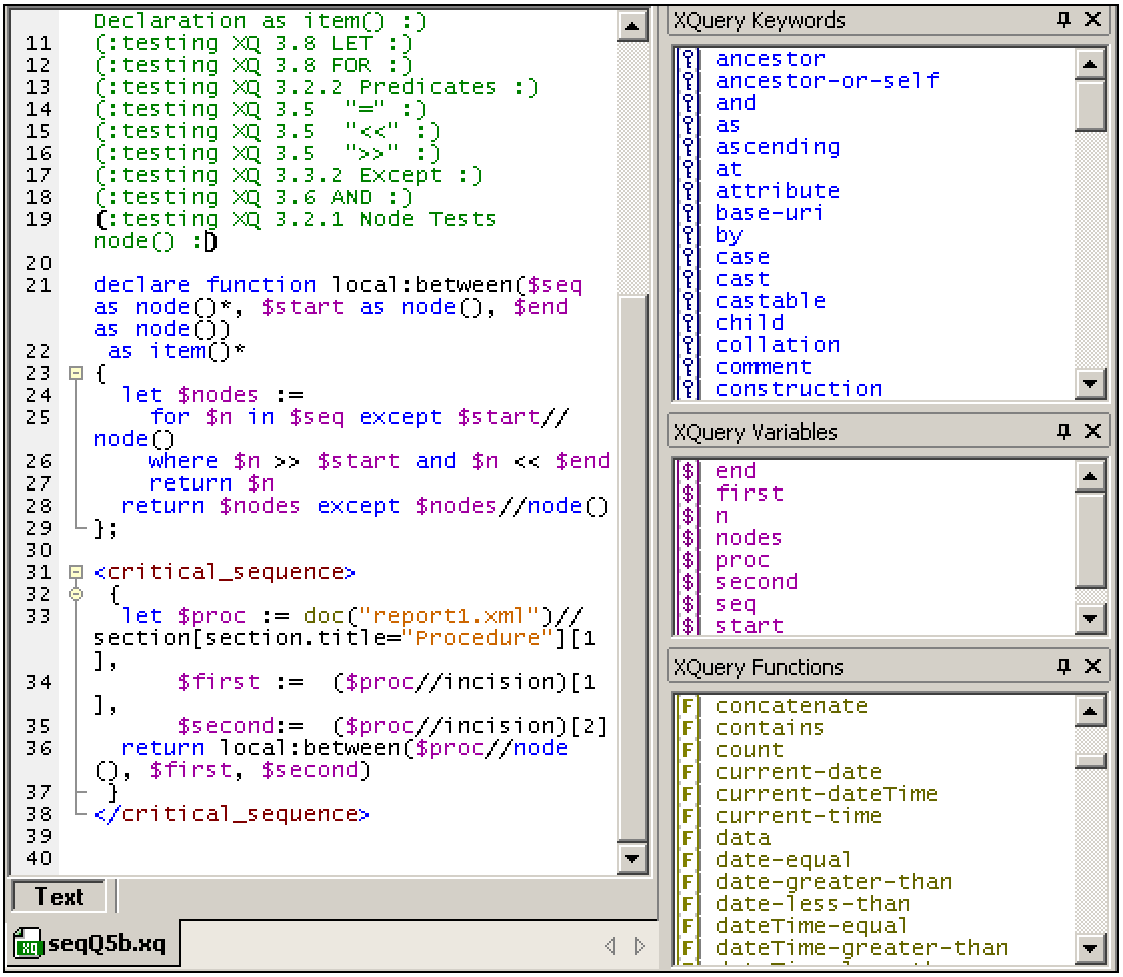
- The color of items in the three Entry Helpers are different and correspond to the syntax coloring used in the text. These colors cannot be changed.

- 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.

3.4.3 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; line numbering and folding margins in Enterprise and Professional Editions only*).



```
Declaration as item() :)
(:testing XQ 3.8 LET :)
(:testing XQ 3.8 FOR :)
(:testing XQ 3.2.2 Predicates :)
(:testing XQ 3.5  "=" :)
(:testing XQ 3.5  "<<" :)
(:testing XQ 3.5  ">>" :)
(:testing XQ 3.3.2 Except :)
(:testing XQ 3.6 AND :)
(:testing XQ 3.2.1 Node Tests
node() :)

declare function local:between($seq
as node()*, $start as node(), $end
as node())
 as item()*
{
  let $nodes :=
    for $n in $seq except $start//
node()
    where $n >> $start and $n << $end
    return $n
  return $nodes except $nodes//node()
};

<critical_sequence>
 {
  let $proc := doc("report1.xml")//
section[section.title="Procedure"][1
],
      $first := ($proc//incision)[1
],
      $second:= ($proc//incision)[2]
  return local:between($proc//node
(), $first, $second)
 }
</critical_sequence>
```

XQuery Keywords

ancestor
ancestor-or-self
and
as
ascending
at
attribute
base-uri
by
case
cast
castable
child
collation
comment
construction

XQuery Variables

end
first
n
nodes
proc
second
seq
start

XQuery Functions

concatenate
contains
count
current-date
current-dateTime
current-time
data
date-equal
date-greater-than
date-less-than
dateTime-equal
dateTime-greater-than

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

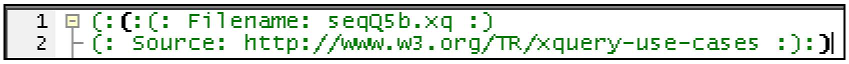
You can change these default colors and other font properties in the Text Fonts tab of the Options dialog (**Tools | Options**).

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).

3.4.4 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*).



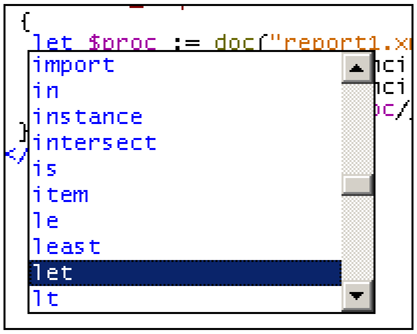
```
(:(:(: Filename: seqQ5b.xq :)
(: Source: http://www.w3.org/TR/xquery-use-cases :):)
```

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 inside a start or end tag of an XML element, clicking **Ctrl+E** highlights the other member of the pair. Clicking **Ctrl+E** repeatedly enables you to switch between the start and end tags. This is another aid to locating the start and end tags of an XML element.

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.

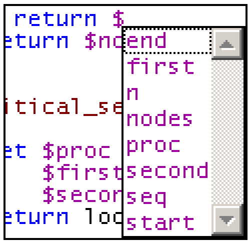


```
{
 let $proc := doc("report1.x
```

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.)



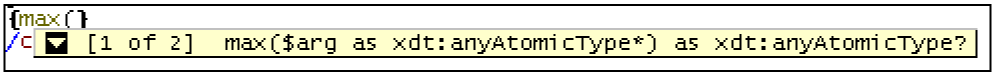
To insert a variable after the **$** character (when typing), or to replace a selected variable, double-click the variable you want in the pop-up menu.

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.



```
{max()
/c ▼ [1 of 2]  max($arg as xdt:anyAtomicType*) as xdt:anyAtomicType?
```

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.

3.4.5 Validation and Execution of XQuery Documents

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.

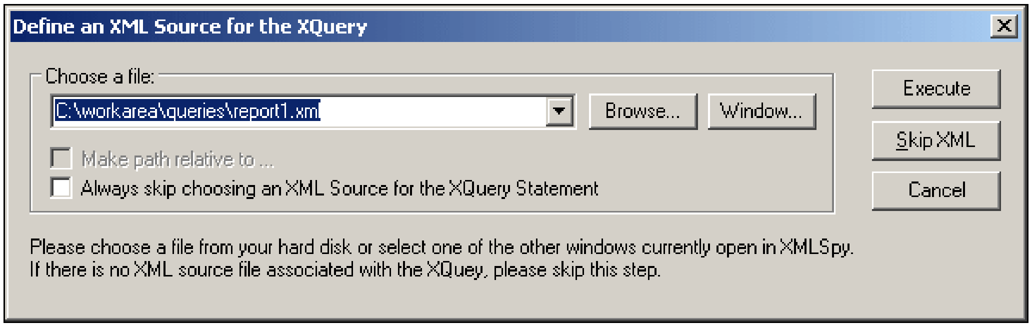
Executing XQuery documents

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.



Choose a file: C:\workarea\queries\report1.xml

□ Make path relative to ...

□ Always skip choosing an XML Source for the XQuery Statement

Please choose a file from your hard disk or select one of the other windows currently open in XMLSpy.
If there is no XML source file associated with the XQuey, please skip this step.

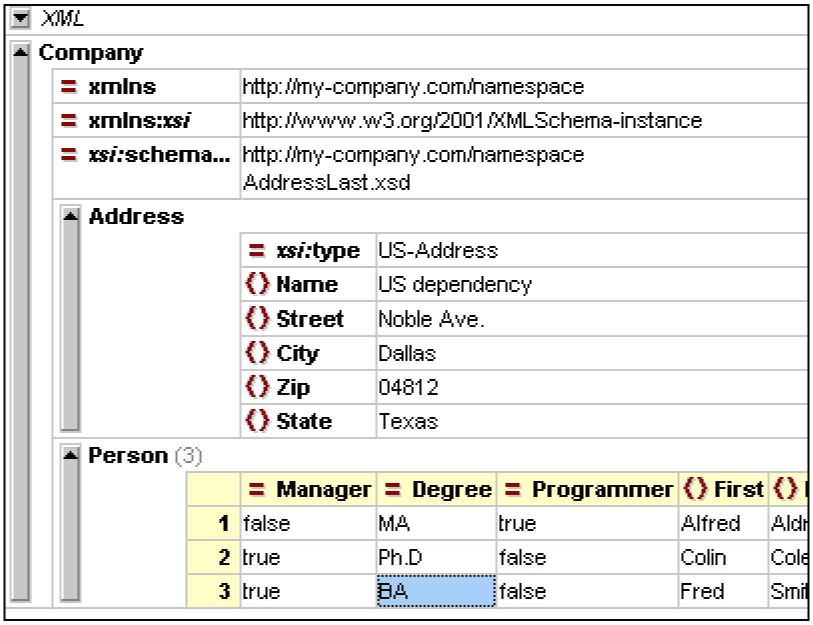
3. Do one of the following:
   - To select an XML file, use either the **Browse** button or the **Window** button (which lists files that are open in XMLSpy and that are in XMLSpy projects). Select an XML source if you wish to assign its document node as the context item for the root level of the XQuery 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.

For details about how the Altova XQuery Engine is implemented and will process XQuery files, see XQuery Engine Implementation.

4 Enhanced Grid View

Clicking the **Grid** button at the bottom of the Document Window opens the Enhanced Grid View (or Grid View). The Enhanced Grid View shows the hierarchical structure of XML documents 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	() I
1	false	MA	true	Alfred	Aldr
2	true	Ph.D	false	Colin	Cole
3	true	BA	false	Fred	Smi

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 the 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 | Enhanced 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

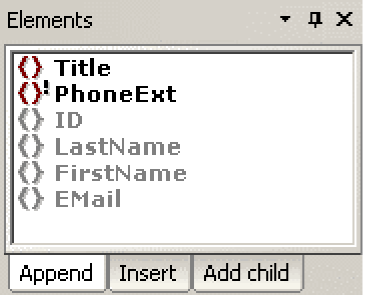
Grid View enables intelligent editing when the XML document is based on a schema. See Editing in Grid View for details. The Entry Helpers used in Grid View are described below.

Embedded Database/Table view

Grid View allows you to display recurring elements in a Database/Table View. This function is available for any type of XML file: XML, XSD, XSLT, etc. The Entry Helpers used in Database/Table View are described below.

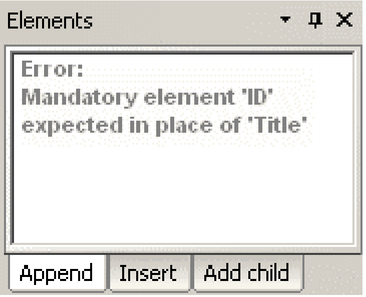
Elements Entry Helper

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.

If you create a structure that does not match the content model specified in your schema, the built-in validating parser displays an error message in the Elements Entry Helper window in Grid View.

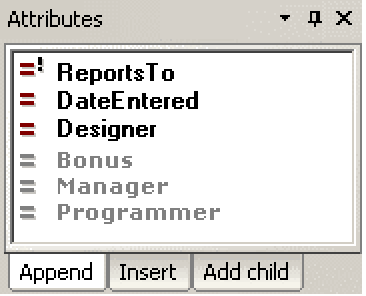


Error:
Mandatory element 'ID' expected in place of 'Title'

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.

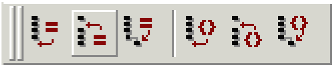
Please note: If you add an internal entity, you will need to save and reopen your document before the entity appears in the Entities Entry Helper.

4.1 Editing in Grid View

When editing an XML document based on a schema (DTD or XML Schema), Enhanced 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.



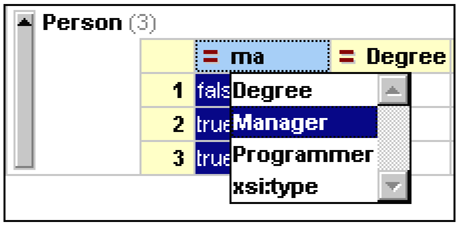
To insert or append an element or attribute:

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. 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.

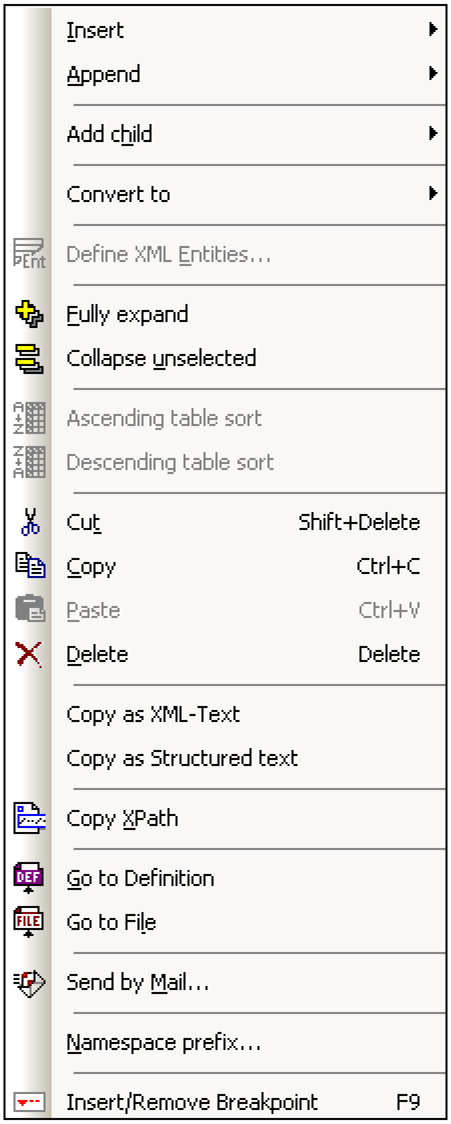


To edit an element or attribute name:

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.)

Grid View context menu

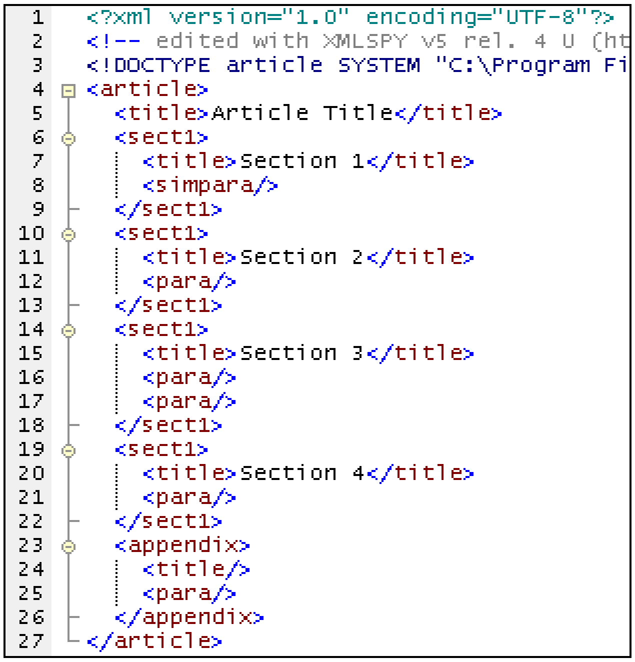
In addition to the commands available through the various menus and toolbars, Grid View also provides a context menu (shown below) that contains the most frequently used commands collected from several menus in one convenient place. To open the context menu, right-click the item for which you wish to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


Insert
Append
Add child
Convert to
Define XML Entities...
Fully expand
Collapse unselected
Ascending table sort
Descending table sort
Cut Shift+Delete
Copy Ctrl+C
Paste Ctrl+V
Delete Delete
Copy as XML-Text
Copy as Structured text
Copy XPath
Go to Definition
Go to File
Send by Mail...
Namespace prefix...
Insert/Remove Breakpoint F9

4.2 Database/Table View

The **Database/Table View** (hereafter Table View) is integrated in the Enhanced 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.

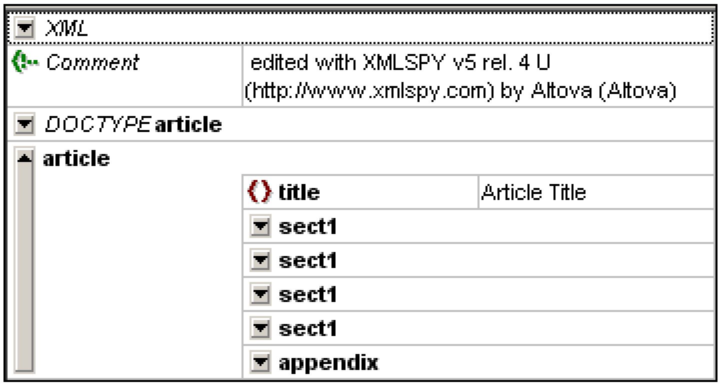
For instance, consider the following XML document:



```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v5 rel. 4 U (ht
<!DOCTYPE article SYSTEM "C:\Program Fi
<article>
  <title>Article Title</title>
  <sect1>
    <title>Section 1</title>
    <simpara/>
  </sect1>
  <sect1>
    <title>Section 2</title>
    <para/>
  </sect1>
  <sect1>
    <title>Section 3</title>
    <para/>
    <para/>
  </sect1>
  <sect1>
    <title>Section 4</title>
    <para/>
  </sect1>
  <appendix>
    <title/>
    <para/>
  </appendix>
</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:



XML

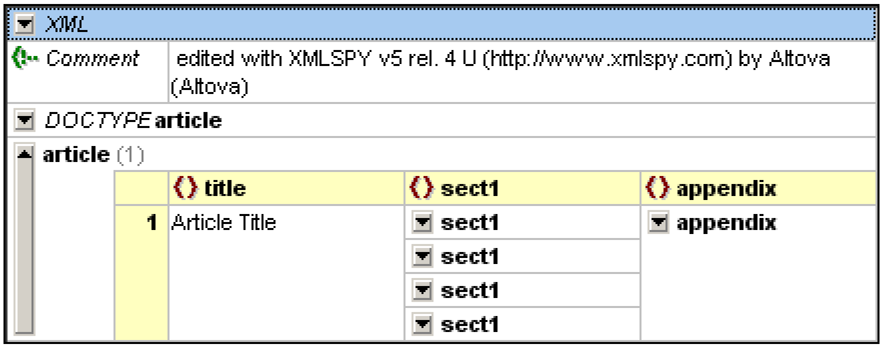
Comment: edited with XMLSPY v5 rel. 4 U (http://www.xmlspy.com) by Altova (Altova)

DOCTYPE article

article

title	Article Title
sect1	
sect1	
sect1	
sect1	
appendix	

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)**.)



XML

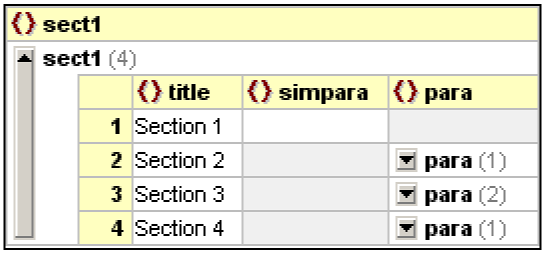
Comment edited with XMLSPY v5 rel. 4 U (http://www.xmlspy.com) by Altova (Altova)

DOCTYPE article

article (1)

	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.



sect1

sect1 (4)

	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.

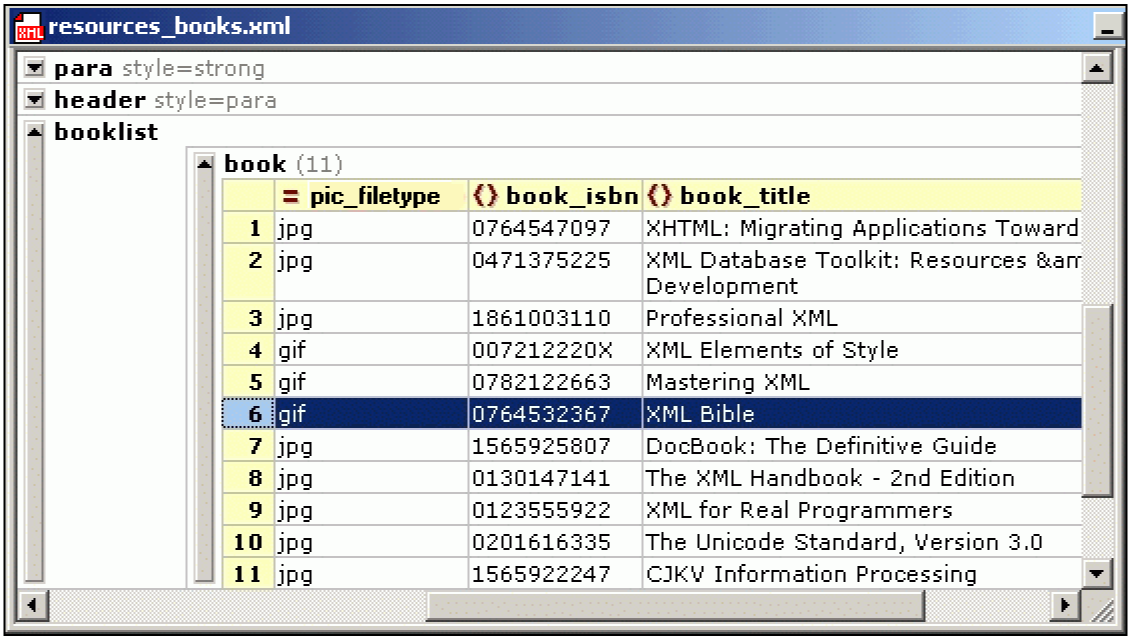
Manipulating table data

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**).

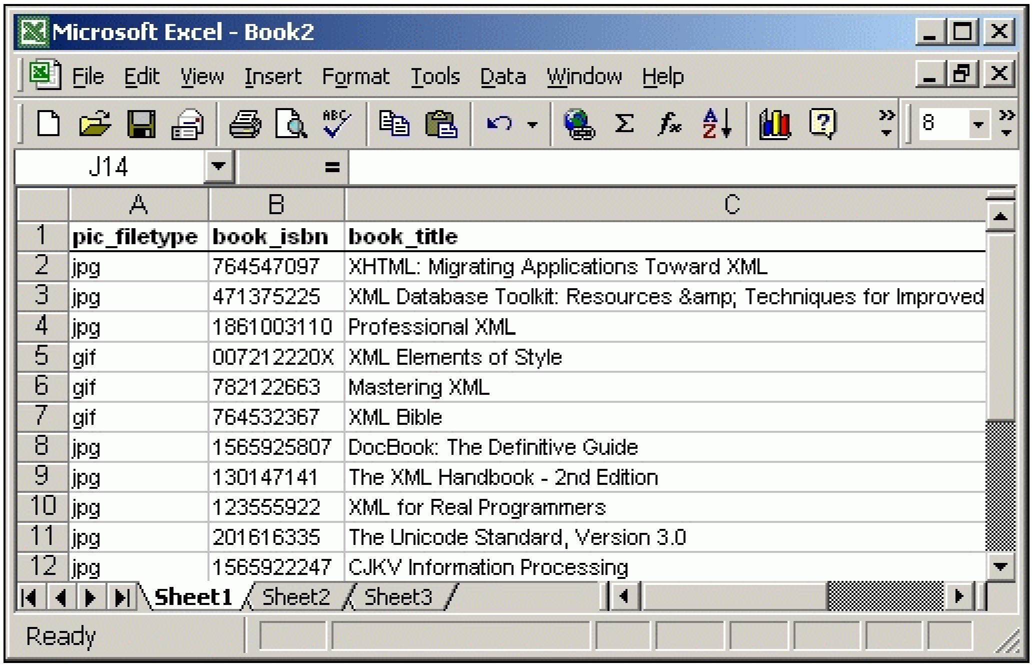
Moving data between Table View and external applications

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 &am 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 &amp; 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.

5 Schema/WSDL Design View

The **Schema/WSDL Design View** is enabled for XML Schema documents. The Schema/WSDL Design View will hereafter be referred to as the Schema Design View, and it is described in detail in the following sub-section. For a description of how to create XML Schemas, see the XML Schema Tutorial. When designing a schema, if you have SchemaAgent installed and licensed on your machine, you can have access not only to the components of the schema currently active in XMLSpy but also to the components of multiple schemas in other repositories on the network. How to set up and work with SchemaAgent is described in the section Working with SchemaAgent.

5.1 Schema Design View

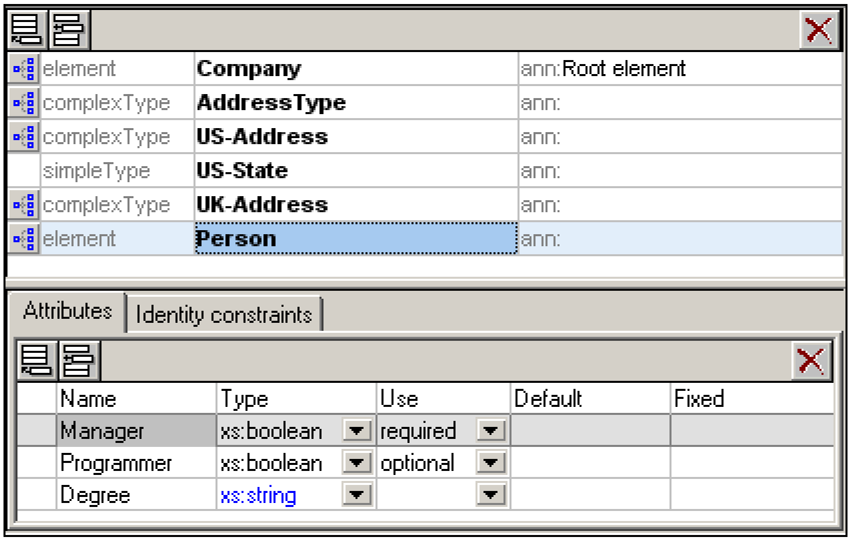
The Schema Design View itself has two types of view:

- A main **Schema Overview**, which displays all global components (global elements, complex types, etc) in a simple table.
- Views of the content models of individual global components (**Content Model View**).

Given below is a brief overview of Schema Overview and Content Model View, followed by a description of the Entry Helpers available in Schema Design View. The two sub-sections of this section contain detailed descriptions of Schema Overview and Content Model View.

Schema Overview

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



element	**Company**	ann:Root element
complexType	**AddressType**	ann:
complexType	**US-Address**	ann:
simpleType	**US-State**	ann:
complexType	**UK-Address**	ann:
element	**Person**	ann:

Attributes | Identity constraints

Name	Type	Use	Default	Fixed
Manager	xs:boolean	required		
Programmer	xs:boolean	optional		
Degree	xs:string			

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 the 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).

Note the following editing features of Schema Overview:

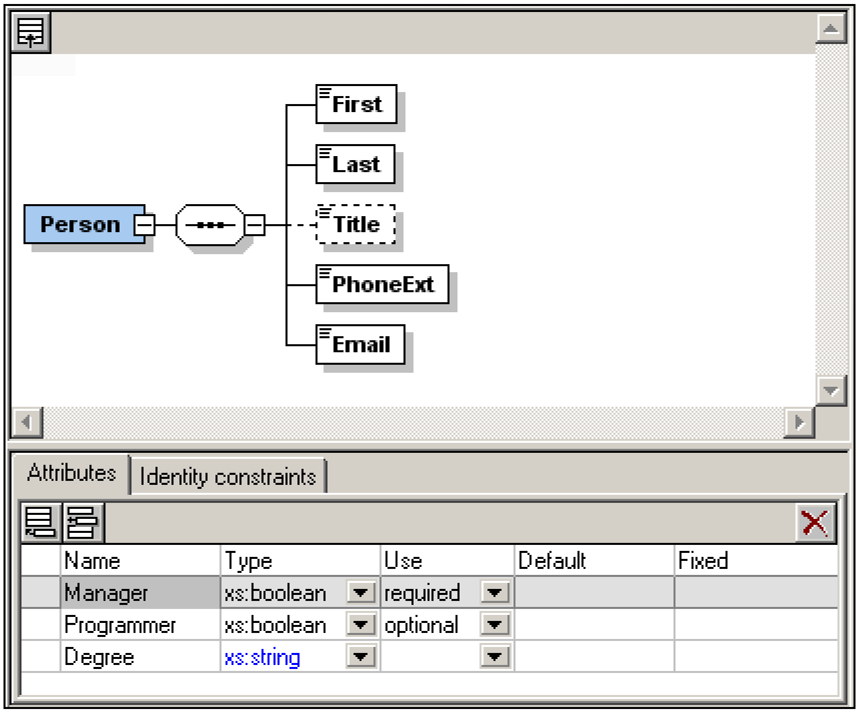
- You can reposition components in the Schema Overview list using drag-and-drop.
- You can navigate using the arrow keys of your keyboard.
- You can copy or move global components, attributes, and identity constraints to a different position and from one schema to another using cut/copy-and-paste.
-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

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 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**).



Name	Type	Use	Default	Fixed
Manager	xs:boolean	required		
Programmer	xs:boolean	optional		
Degree	xs:string			

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 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.

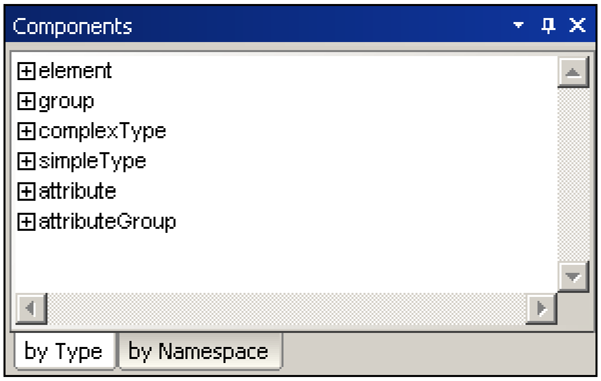
To return to the Schema Overview, click the **Show Globals** icon or select the menu option **Schema design | Display All Globals**.

Entry Helpers in Schema View

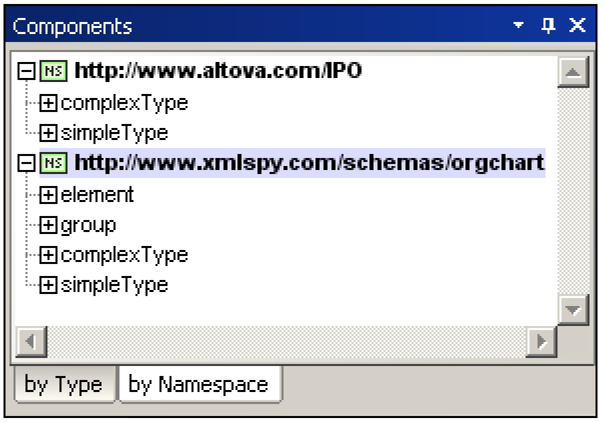
There are three Entry Helpers in Schema/WSDL Design View: Component Navigator, Details Entry Helper, and Facets Entry Helper. These are described below.

Component Navigator
The Component Navigator is an Entry Helper in **Schema/WSDL Design View**. It serves two 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.



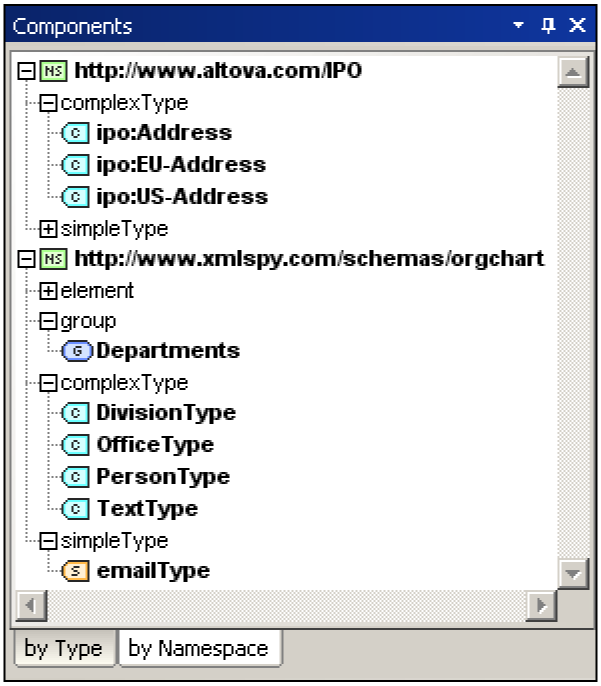
In the Type tab *(above)* global components are grouped in a tree according to their component type. In the Namespace tab *(below)*, components are organized first according to namespace and then according to component type. Note that a component type is listed in a tree only if at least one component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

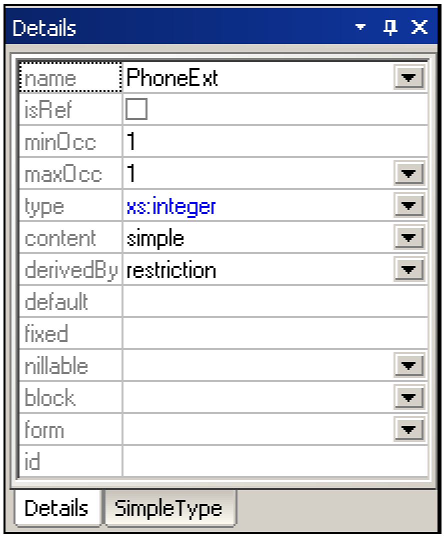


If a component has a content model (i.e., if it is an Element, Group, or Complex Type), double-clicking it will cause the content model of that component to be displayed in Content Model View (in the Main Window). If the component does not have a content model (i.e. if it is a Simple Type, Attribute, 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/WSDL Design 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. Currently supported databases are: Oracle, SQL Server, and Tamino.



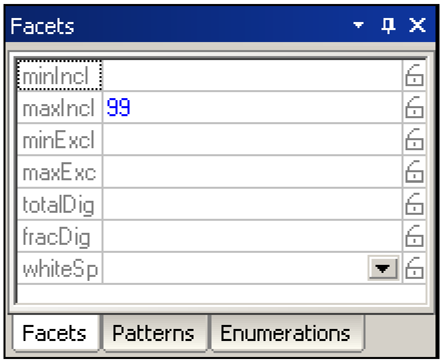
name	PhoneExt
isRef	□
minOcc	1
maxOcc	1
type	xs:integer
content	simple
derivedBy	restriction
default	
fixed	
nillable	
block	
form	
id	

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/WSDL Design View**, and enables you to enter the values of facets, patterns, and enumerations. For examples of how to use facets in an XML Schema, please see the relevant sections in the tutorial.



minIncl	
maxIncl	99
minExcl	
maxExc	
totalDig	
fracDig	
whiteSp	

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

1. Select the required tab (Facets, Patterns, 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.

Please 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.

5.1.1 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

We call these components at the top level **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.

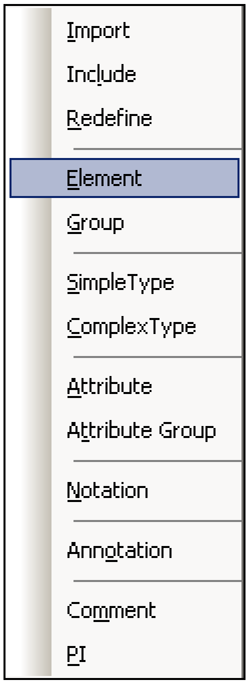
Key terms

- *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).

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.

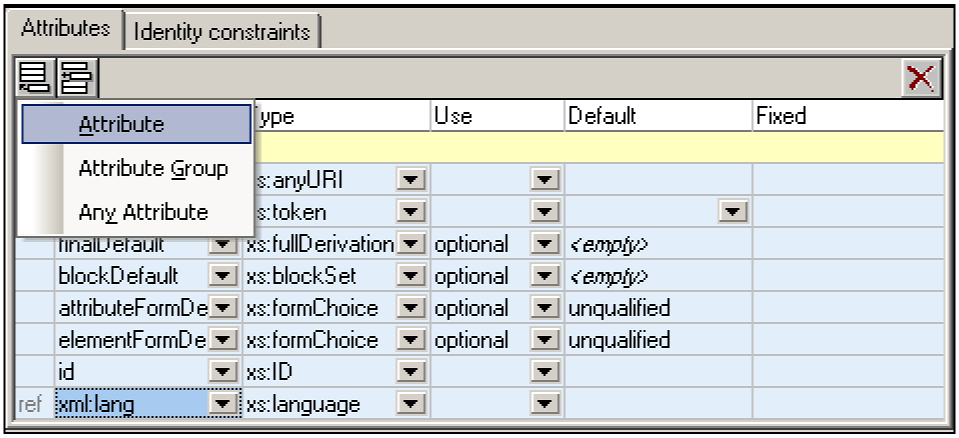
Attributes and identity constraints of components

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.



		ype	Use	Default	Fixed
		s:anyURI			
		s: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	xml:lang	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.

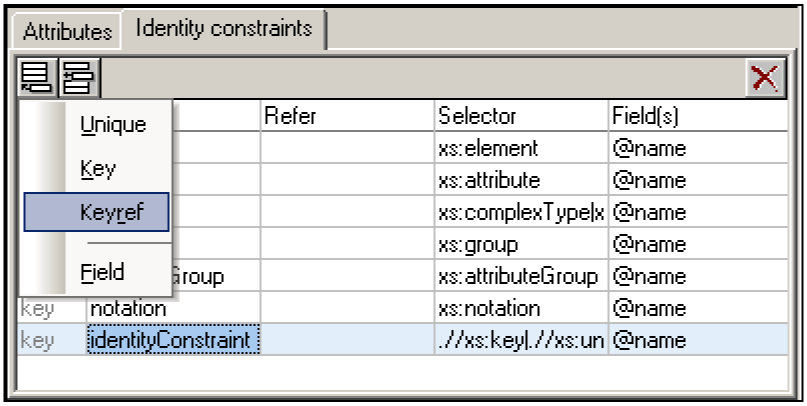
Please note: You can also define attributes for global components in Content Model View: Select the global component, and then define attributes as described above.

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.

To define identity constraints for a component:

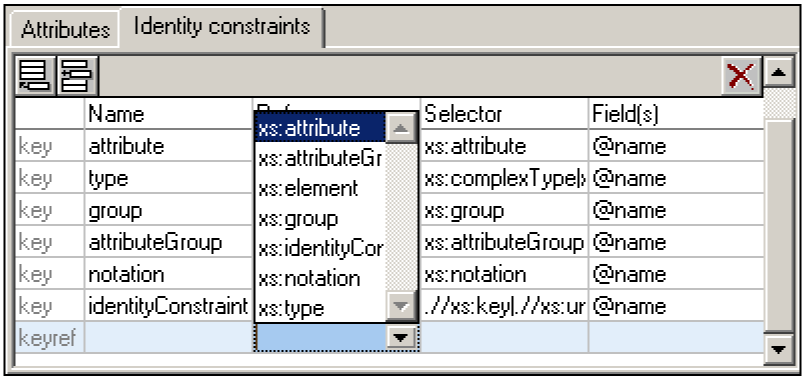
1. Select the global component in the global components list.
2. In the Attributes/Identity Constraints pane, select the Identity Constraints tab.
3. Click the Append or Insert icon at the top left of the Identity Constraints tab.
4. From the popup that appears, select the type of identity constraint you want to append or insert.



		Refer	Selector	Field(s)
			xs:element	@name
			xs:attribute	@name
			xs:complexType|x	@name
			xs:group	@name
	roup		xs:attributeGroup	@name
key	notation		xs:notation	@name
key	identityConstraint		.//xs:key|.//xs:un	@name

An entry is created in the Identity Constraints list.

5. In the newly created entry, enter the Identity Constraint's properties. `Selector` identifies the nodeset to which the identity constraint will apply. `Field` identifies the value which must be unique within the nodeset identified by the `Selector` property. If you have selected `Keyref` as the identity constraint type, the `Refer` property is enabled, and the combo-box in it lists all Keys that have been defined in the schema (*see screenshot below*).



	Name		Selector	Field(s)
key	attribute		xs:attribute	@name
key	type		xs:complexType|x	@name
key	group		xs:group	@name
key	attributeGroup		xs:attributeGroup	@name
key	notation		xs:notation	@name
key	identityConstraint		.//xs:key|.//xs:ur	@name
keyref				

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

5.1.2 Content Model View

The **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:

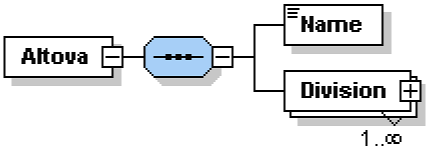
`sequence`, `choice`, and `all`.

To insert a compositor:

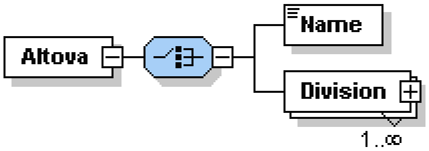
1. Right-click the element to which you wish to add child elements
2. Select **Add Child | Sequence** (or **Choice** or **All**).

The compositor is added, and will look as below:

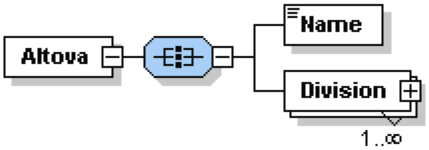
- **Sequence**



- **Choice**



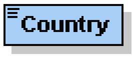
- **All**



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.

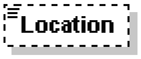
Components in the Content Model
Given below is a list of components that are used in content models. The graphical representation of each provides detailed information about the component's type and structural properties.

- 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.

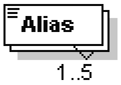
- Single optional element



Details: The rectangle indicates an element and the dashed border means the element

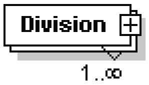
is optional. The absence of a number range indicates a single element (i.e. `minOcc=0` and `maxOcc=1`). Element name is `Location`.
Note: The context menu option **Optional** converts a mandatory element into an optional one.

- Mandatory multiple element

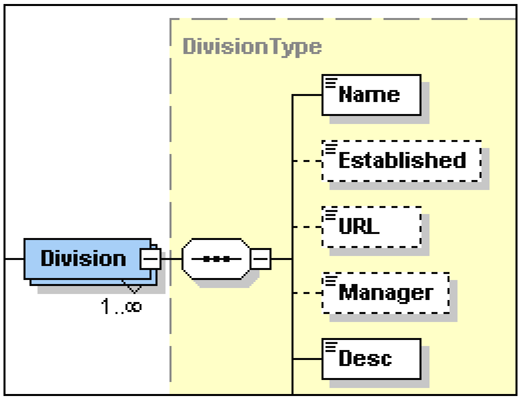


Details: The rectangle indicates an element and the solid border indicates that the element is required. The number range 1..5 means that `minOcc=1` and `maxOcc=5`. Element name is `Alias`.

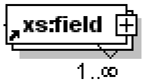
- Mandatory multiple element containing child elements



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.



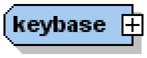
- Element referencing global element



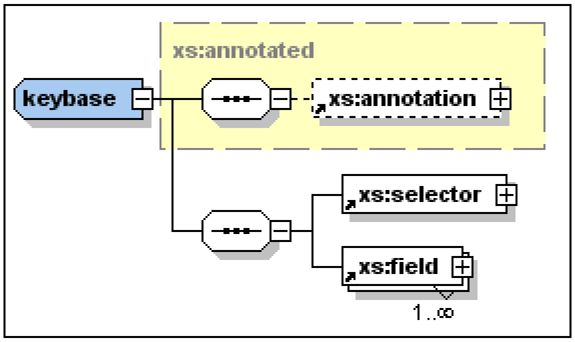
Details: The arrow in the bottom-left means the element references a global element. The rectangle indicates an element and the solid border indicates that the element is required. The number range 1..infinity means that `minOcc=1` and `maxOcc=unbounded`. The plus sign indicates complex content (i.e.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

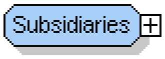


The `keybase` complex type shown above was declared in Schema Overview with a base type of `xs:annotated`. The base type is displayed as a rectangle with a dashed gray border and a yellow background color. Then, in Content Model View, the child elements `xs:selector` and `xs:field` were created. (Note 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.

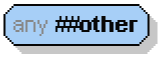
Please note: The base type of a content model is displayed as a rectangle with a dashed gray border and a yellow background color. You can go to the content model of the base type by double-clicking its name.

- Model group



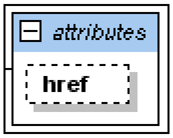
Details: The irregular octagon with a plus sign indicates a model group. A model group allows you to define and reuse element declarations.
Note: When the model group is declared (in Schema Overview) it is given a name. You subsequently define its content mode (in Content Model View) by assigning it 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



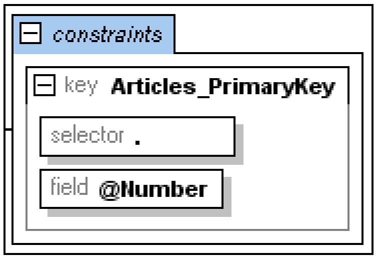
Details: The irregular octagon with `any` at left indicates a wildcard.
Note: Wildcards are used as placeholders to allow elements not specified in the schema or from other namespaces. `##other` = elements can belong to any namespace other than the target namespace defined in the schema; `##any` = elements 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 dashed border.
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 in Diagram 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 in the Content Model View or in a pane below the Content Model View. You can toggle between these two views by clicking the Display in Diagram icon.

Please note:

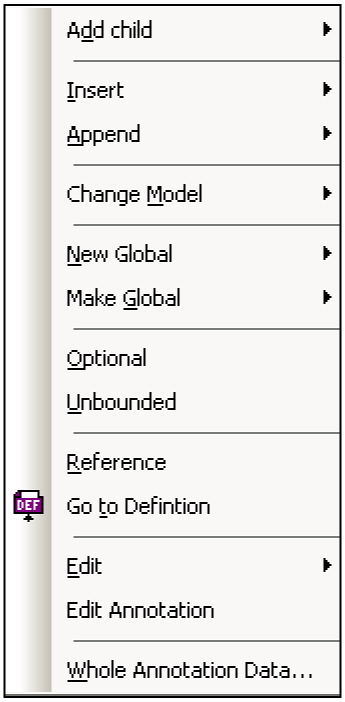
- Property descriptor lines you have defined in the Schema Display Configuration dialog can be turned on and off by clicking the Add Predefined Details toolbar icon.
- You can toggle Attributes and Identity Constraints to appear either in the diagram of the content model itself or in a pane below the Content Model View by clicking the Display

in Diagram icon.

- 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 Context 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

1. Right-click the element.
2. Select the context menu option **Edit | Name** and edit the name.

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.

session_date

Date and time when interview was held

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.

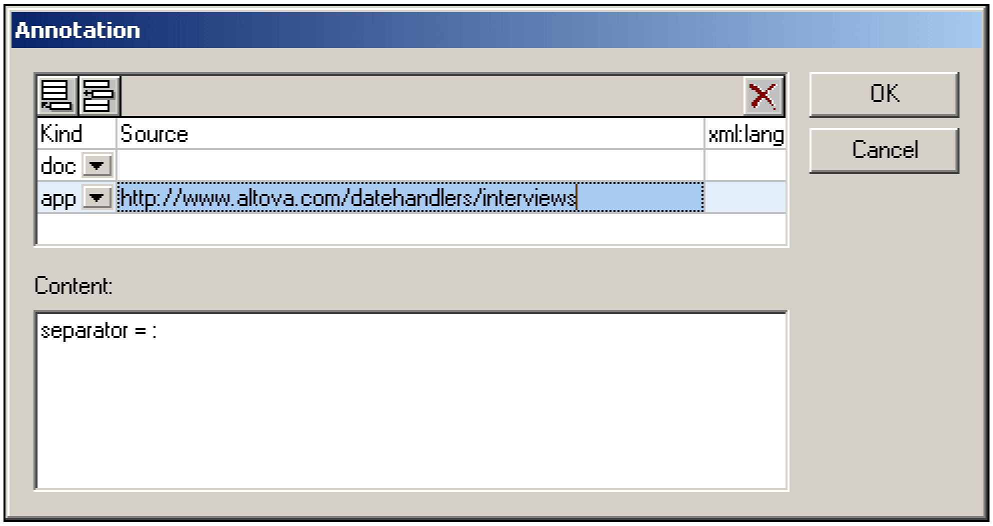
Creating and editing application info for a compositor or component

1. Right-click the compositor or component.

session_date

Date and time when interview was held

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.



Kind	Source	xml:lang
doc		
app	http://www.altova.com/datehandlers/interviews	

Content:

separator = :

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.

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>
```

Configuring the content model view

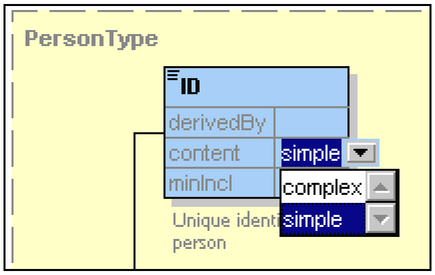
You can configure the content model view for the entire schema in the Schema display configuration dialog (**Schema design | Configure view**). For details about configuration options, see the Configure view section later in the User Reference. Note that the settings you define here apply to the whole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

5.1.3 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.

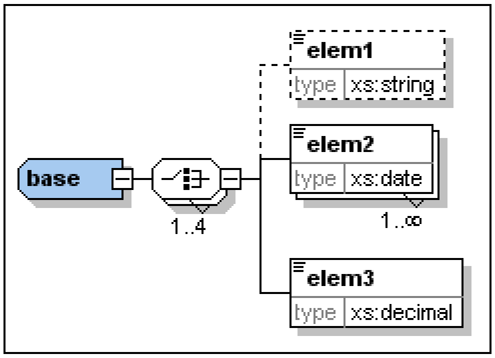
Smart Restrictions combine and correlate the two content models in the graphical view of the derived content model. In the derived complex type, all particles of the base complex type, and how they relate to the derived type, can be seen. Additionally, Smart Restrictions provide visual hints to show 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 RES.

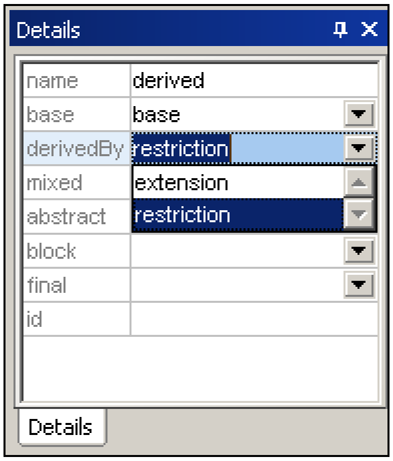
The example that follows illustrates the features of Smart Restrictions.

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



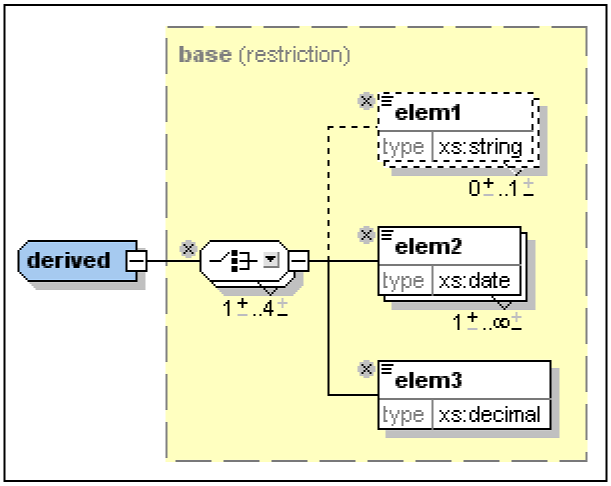
The complex type "derived" is derived from the "base" type as follows:

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.



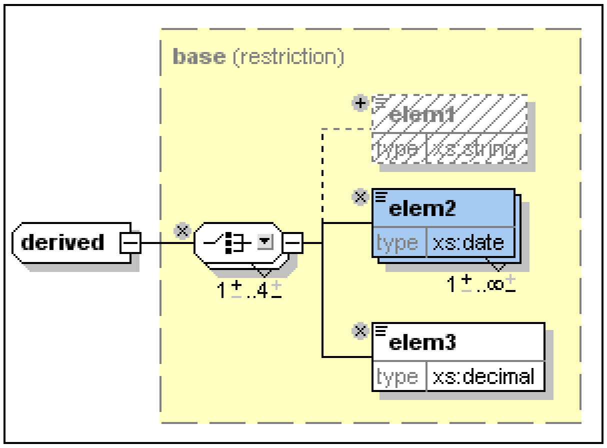
name: derived
base: base
derivedBy: restriction

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

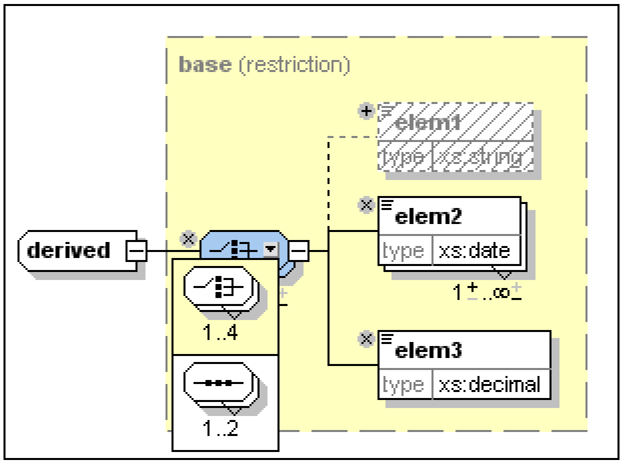


Notice the following controls that can be used to restrict the derived type in this example:

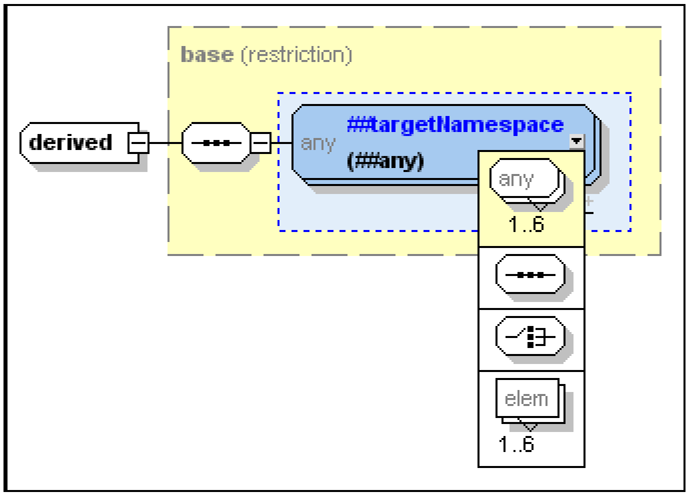
- Use this icon ⊗ to remove elements that are in the base type from the derived type. Here, elem1 has been deleted. To add it again, click this icon ⊕.



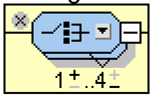
- Click the down arrow on the Choice element ▼ to get the following list, which allows you to change the Choice model group to a Sequence model group:

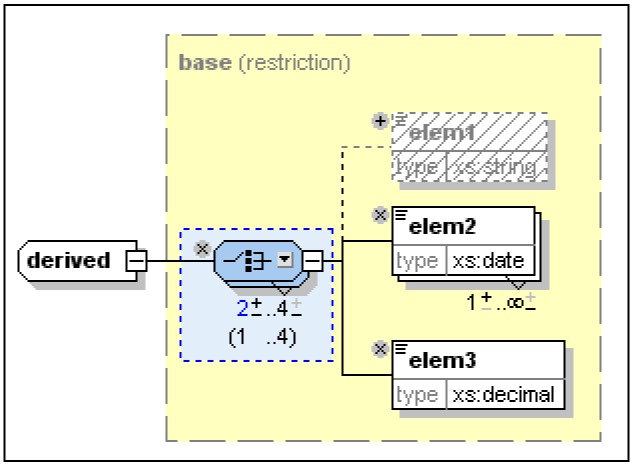


It is also possible to change wildcards in the same way, as seen in this example:



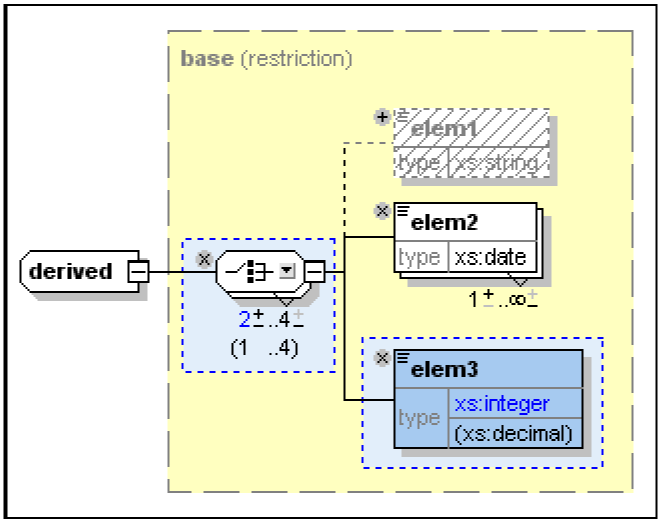
For a complete list of which particles can be replaced by which other particles, see the XML schema specification.

- 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.

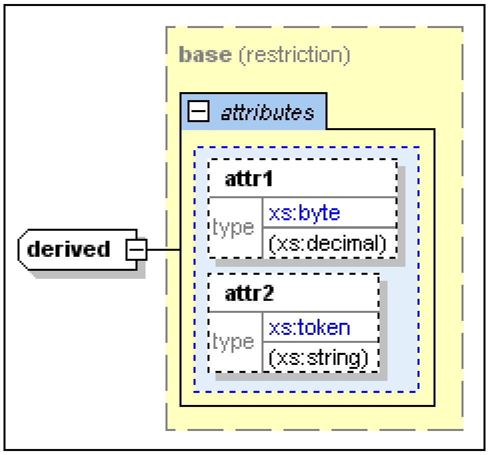


Here you can see that the minimum occurrence for this element has been changed to 2. Notice that the model group now has a blue background, which means that it is no longer the same as the model group in the base complex type. Also, the permitted occurrence range of the model group in the base particle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

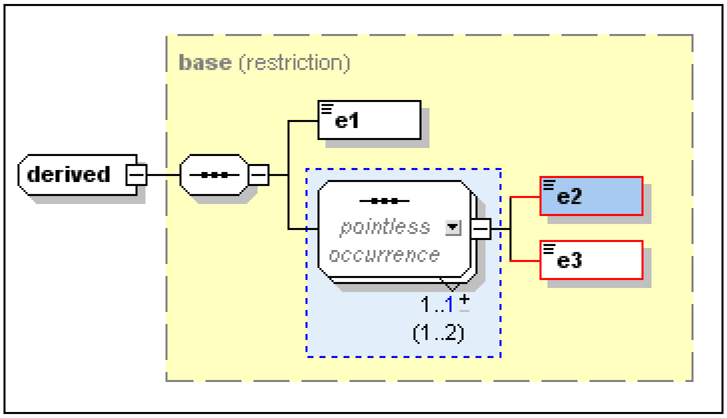
- It is possible to change the data types of attributes or elements if the new data type is a valid restriction of the base data type as defined in the XML schema specification. For example, you can change the data type of elem3 in the "derived" data type from decimal to integer. After you do this, the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

5.1.4 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's Enterprise XML Suite and Professional

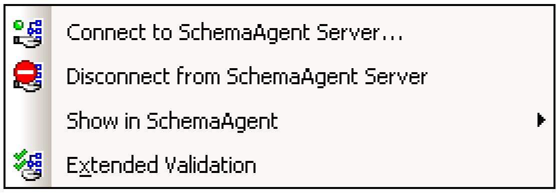
XML Suite products each include the SchemaAgent product and a license key for it. (The SchemaAgent Server application, however, is not included in the XML Suite packages.)

- 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.


Connect to SchemaAgent Server...
Disconnect from SchemaAgent Server
Show in SchemaAgent
Extended Validation

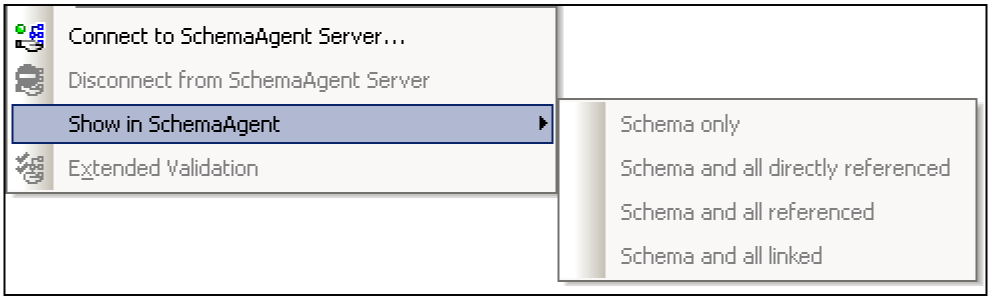
The menu commands provide general administrative functionality. The Entry Helpers (and standard GUI mechanisms, such as drag-and-drop) are used to actually edit schemas.

This section describes how to use the SchemaAgent functionality available in Schema View.

Connecting to SchemaAgent Server

Please note: SchemaAgent Client must be installed in order for you to be able to make a connection.

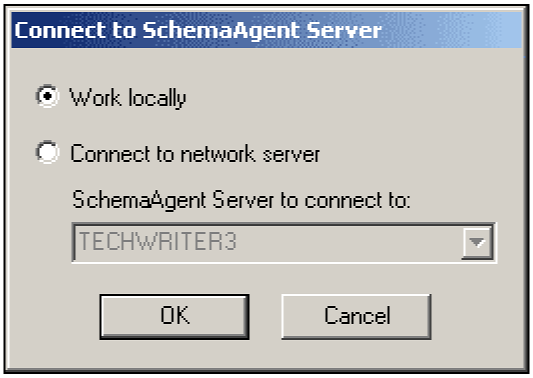
Before you connect to SchemaAgent Server, only the **Connect to SchemaAgent Server** command is enabled in the **Schema Design** menu; other SchemaAgent commands in the **Schema Design** menu are disabled (*see screenshot*). The other menu items become enabled once a connection to a SchemaAgent Server has been successfully made.


Connect to SchemaAgent Server...
Disconnect from SchemaAgent Server
Show in SchemaAgent
Extended Validation
Schema only
Schema and all directly referenced
Schema and all referenced
Schema and all linked

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 (*screenshot below*) opens:



Connect to SchemaAgent Server

● Work locally

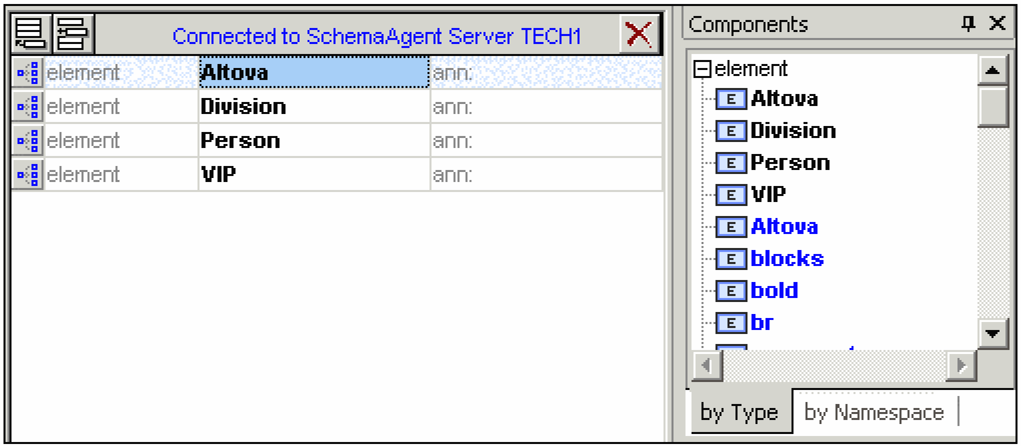
○ Connect to network server

SchemaAgent Server to connect to: TECHWRITER3

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.

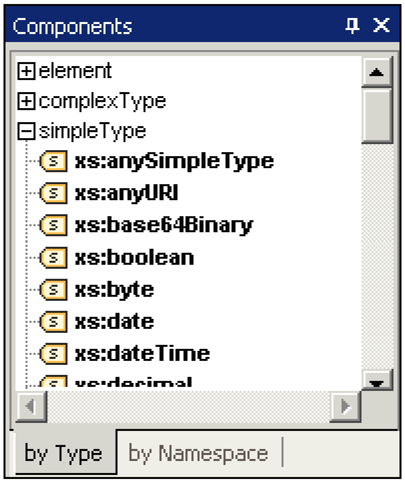
> **Note on servers running with Windows XP SP2**
> If the SchemaAgent Server name is listed in the Connect to SchemaAgent Server dialog but you cannot connect to it, it is possible that your server is not taking part in the name resolution process of your network. Name resolution is blocked by the default 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.

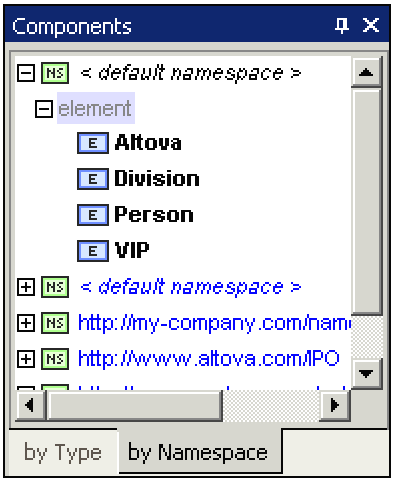
Schema/WSDL View after connecting to SchemaAgent server
After a connection to a SchemaAgent server is established, Schema/WSDL View will look something like this:



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**).



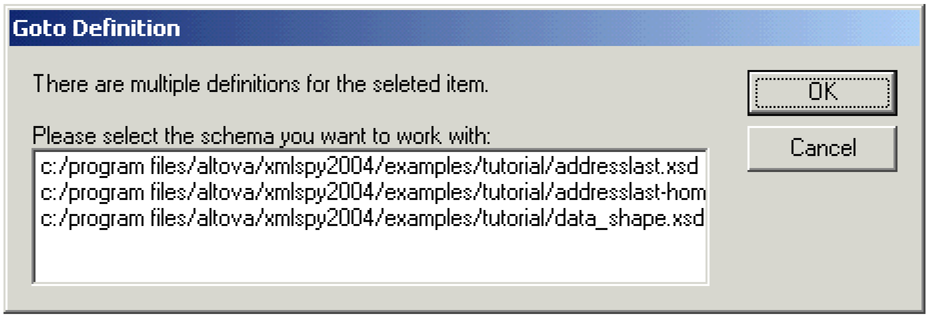


Schema constructs can be viewed by Type, or by Namespace in the respective tabs in the Schema/WSDL View.

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 `By Type` 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.



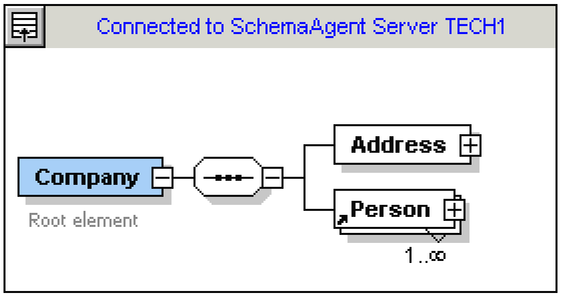
Goto Definition

There are multiple definitions for the seleted item.

Please select the schema you want to work with:

c:/program files/altova/xmlspy2004/examples/tutorial/addresslast.xsd
c:/program files/altova/xmlspy2004/examples/tutorial/addresslast-hom
c:/program files/altova/xmlspy2004/examples/tutorial/data_shape.xsd

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.

Using Schema Constructs

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:

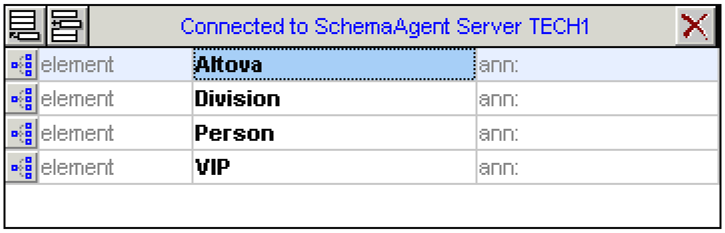
- Global elements can be dragged directly from the Components Entry Helper into the content model of a schema component (in the Schema/WSDL view).
- Components, such as complexTypes and simpleTypes, can be selected from the list box that automatically opens when defining new elements/attributes, etc.
- 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 `By Type` tab of the Components Entry Helper is active.

To use schema constructs from SchemaAgent Server schemas:

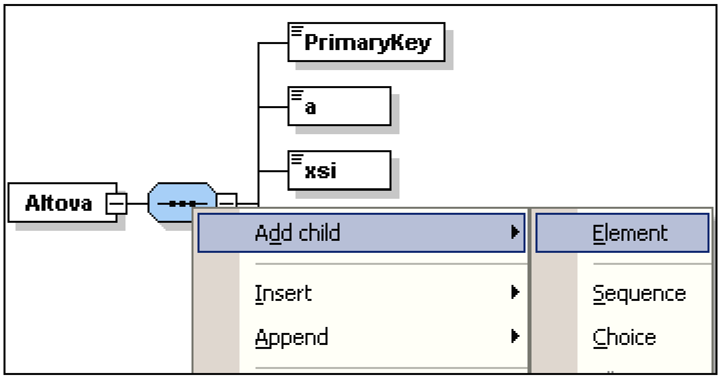
1. Make sure you are connected to a SchemaAgent server (see Connecting to SchemaAgent server).
2. Open and rename the `DB2Schema.xsd` file for this example, for example to `Altova-office`.



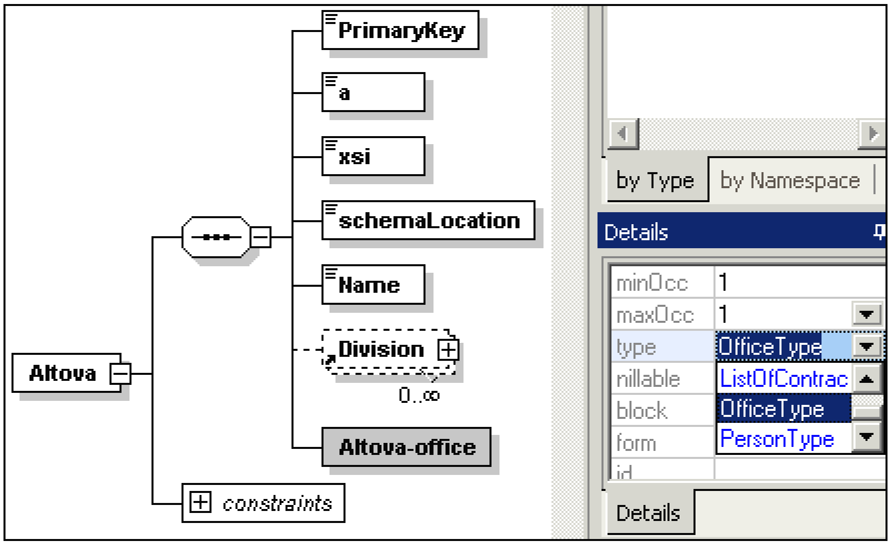
Connected to SchemaAgent Server TECH1

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.

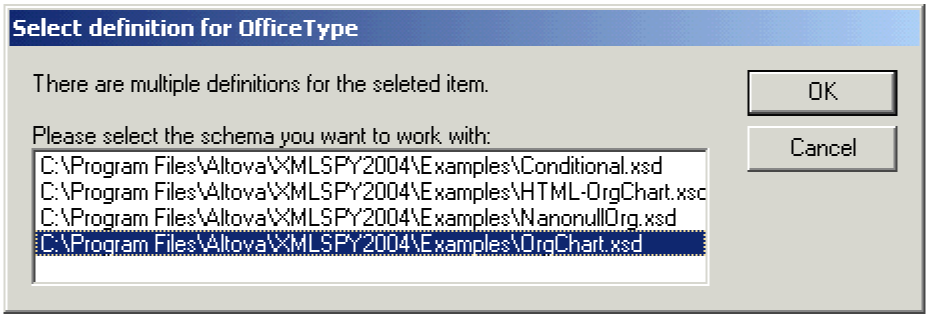


5. Enter `Altova-office` as the name for this new element and press **Enter**.
6. Using the Details Entry Helper, click the `type` combo box and select the entry `OfficeType`.



minOcc	1
maxOcc	1
type	OfficeType
nillable	
block	
form	
id	

This opens the Select Definition For OfficeType dialog box.



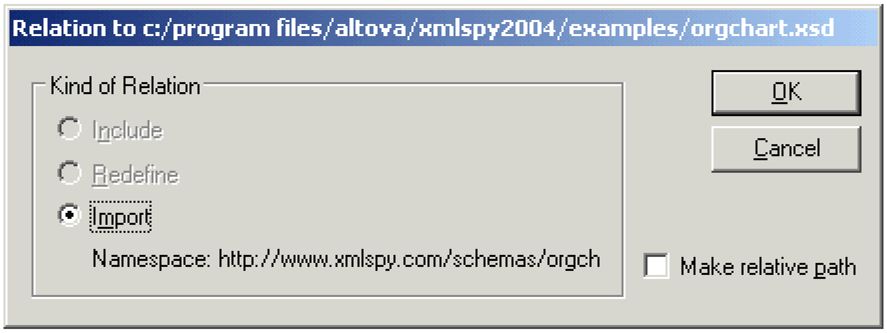
Select definition for OfficeType

There are multiple definitions for the seleted item.

Please select the schema you want to work with:

- C:\Program Files\Altova\XMLSPY2004\Examples\Conditional.xsd
- C:\Program Files\Altova\XMLSPY2004\Examples\HTML-OrgChart.xsc
- C:\Program Files\Altova\XMLSPY2004\Examples\NanonullOrg.xsd
- C:\Program Files\Altova\XMLSPY2004\Examples\OrgChart.xsd

7. Select `Orgchart.xsd` and click **OK** to confirm.



Relation to c:/program files/altova/xmlspy2004/examples/orgchart.xsd

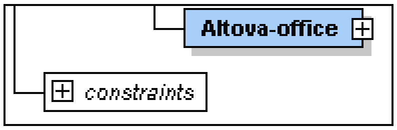
Kind of Relation

○ Include
○ Redefine
● Import

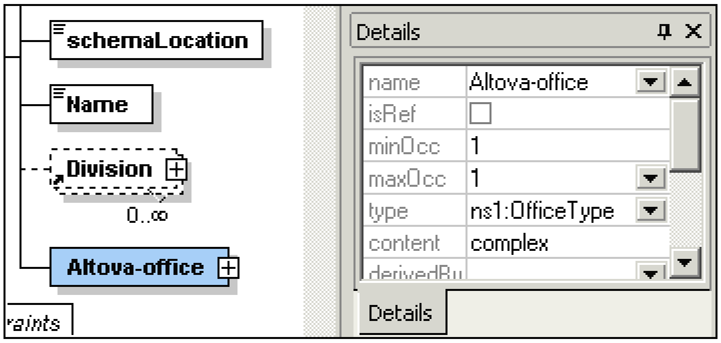
Namespace: http://www.xmlspy.com/schemas/orgch

□ Make relative path

8. Click **OK**. The Import command was automatically selected for you. An expand icon appears in the `Altova-office` element.



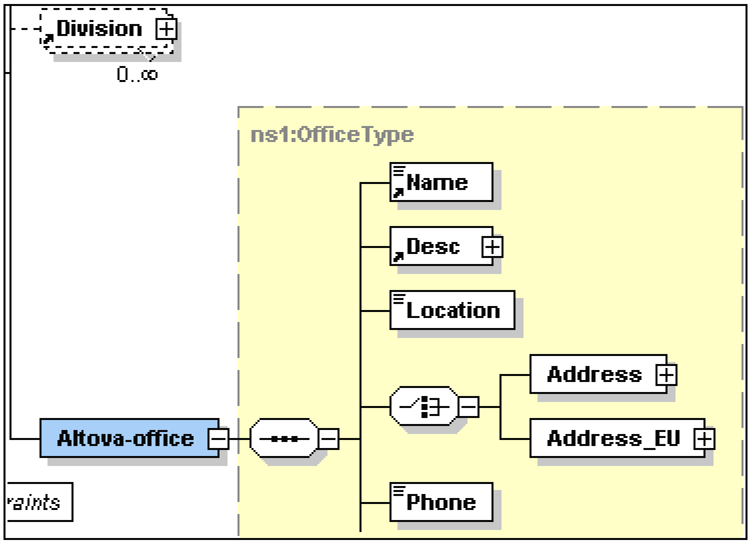
Please note: The `type` entry in the Details entry helper has changed; it is now displayed as `ns1:OfficeType` due to the fact that the `Orgchart.xsd` schema file has been imported and the target namespaces must be different in both schemas. An Import command has also been added to the schema.



Details

name	Altova-office
isRef	□
minOcc	1
maxOcc	1
type	ns1:OfficeType
content	complex
derivedBu	

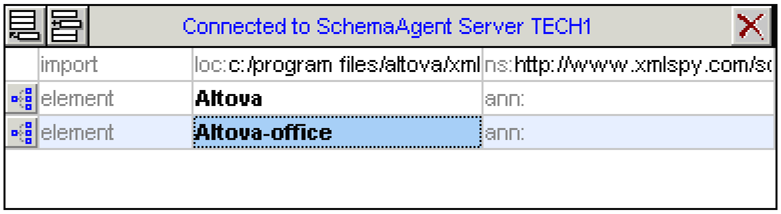
9. Click the Expand button to see the `OfficeType` content model.



10. Press **F8** to validate the schema. The "Schema is valid" message should appear at this stage.

Cleaning up the schema:

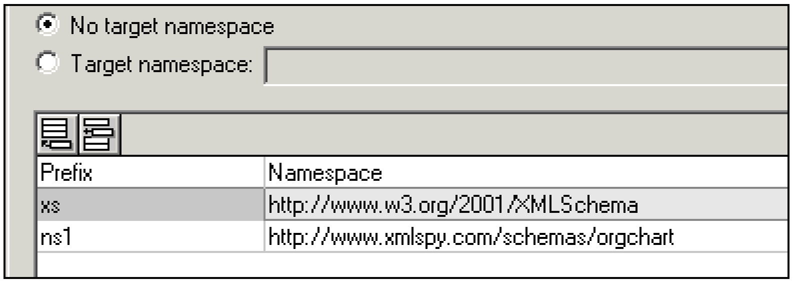
1. Delete the `Division` element in the content model.
2. Click the Return to globals icon to switch to the Schema Overview.
3. Delete the following global elements: `Division`, `Person` and `VIP`.



Connected to SchemaAgent Server TECH1

import	loc:c:/program files/altova/xml	ns:http://www.xmlspy.com/sc
element	**Altova**	ann:
element	**Altova-office**	ann:

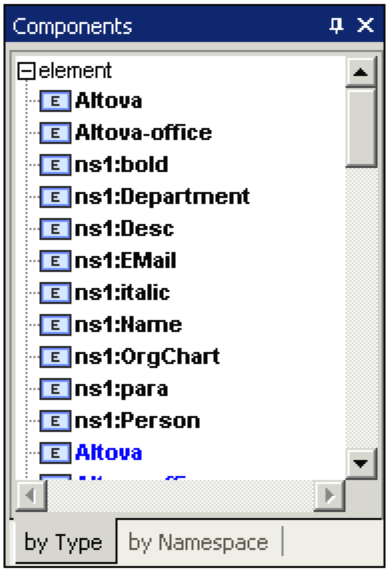
4. Select the menu option **Schema Design | Schema settings** to see how the namespace settings have changed.



● No target namespace
○ Target namespace:

Prefix	Namespace
xs	http://www.w3.org/2001/XMLSchema
ns1	http://www.xmlspy.com/schemas/orgchart

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.

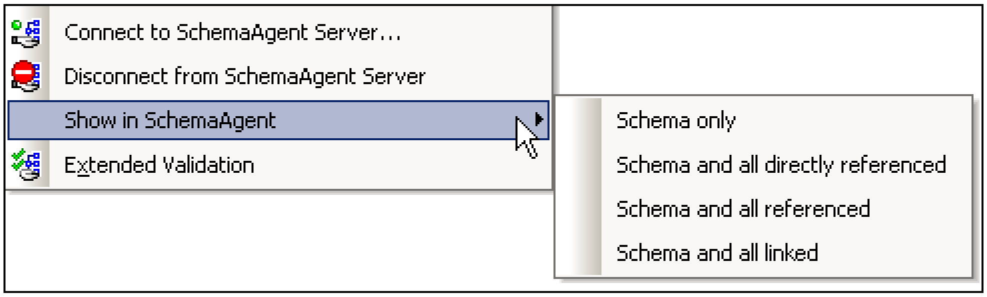


Please note:

- 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.

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).*


Connect to SchemaAgent Server...
Disconnect from SchemaAgent Server
Show in SchemaAgent
Extended Validation
Schema only
Schema and all directly referenced
Schema and all referenced
Schema and all linked

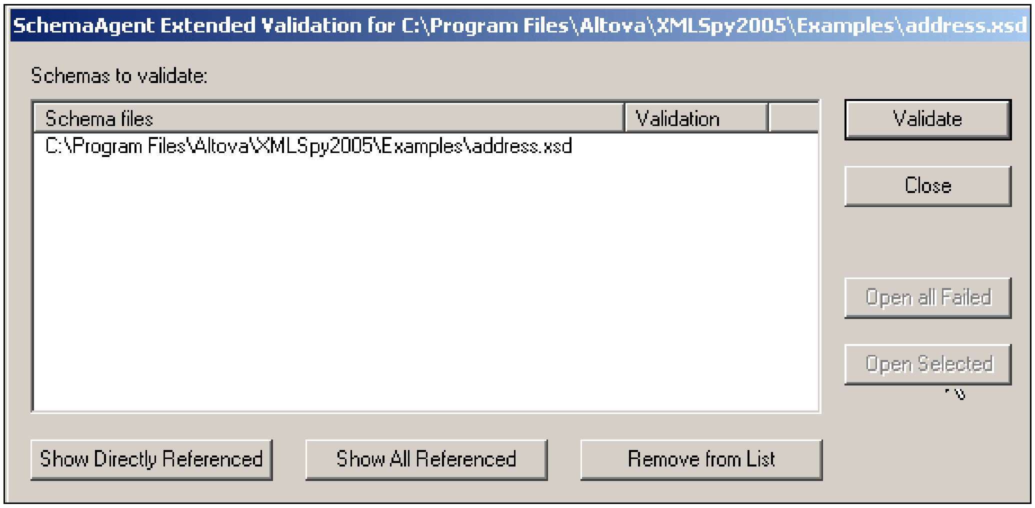
You have the option of opening only the active schema in SchemaAgent, or the active schema together with either (i) all directly referenced schemas, or (ii) all referenced schemas, or (iii) all linked schemas.

Extended 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. There are two types of related schemas that SchemaAgent distinguishes for extended validation: (i) directly referenced schemas, and (ii) referenced schemas.

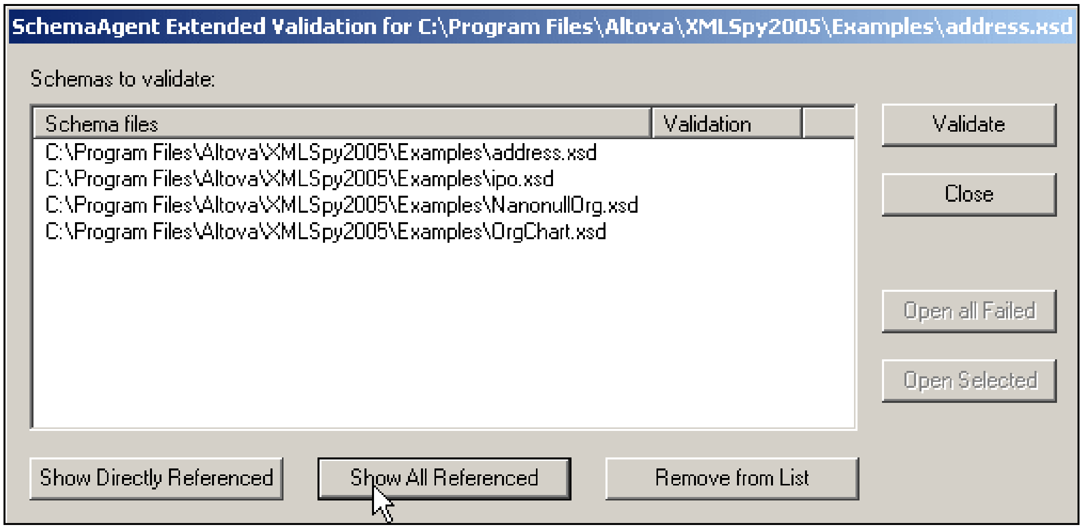
How to carry out extended validation is demonstrated below by means of an example. This example assumes that the schema file `address.xsd` is the active schema in Schema/WSDL View of XMLSpy. You would then do the following:

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.



C:\Program Files\Altova\XMLSpy2005\Examples\address.xsd

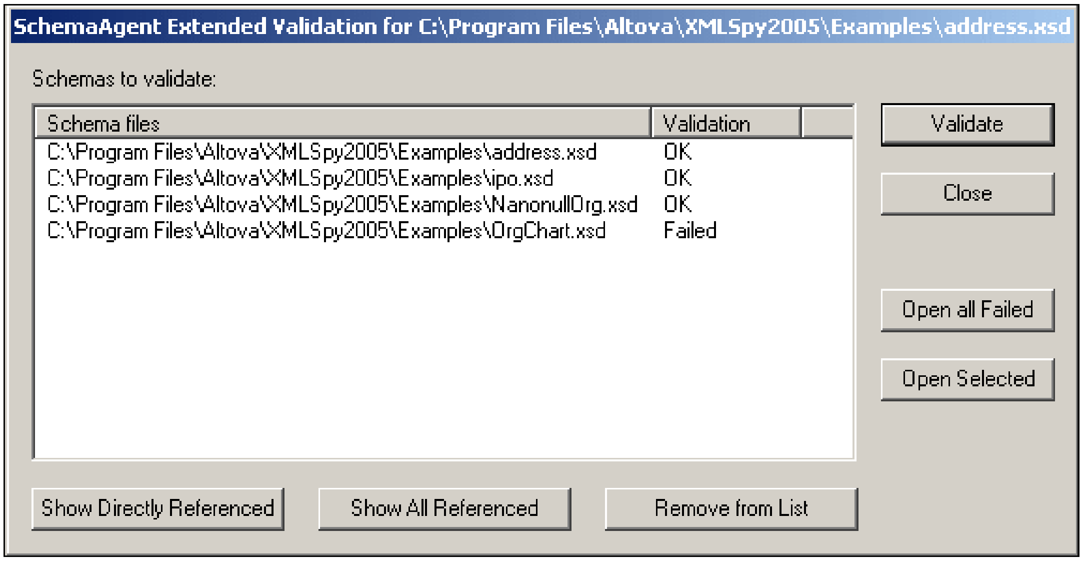
2. To insert schemas into the list, click the **Show Directly Referenced** or **Show All Referenced** button as required. In this example, we have clicked the **Show All Referenced** button, and this inserts all referenced files into the list.



C:\Program Files\Altova\XMLSpy2005\Examples\address.xsd
C:\Program Files\Altova\XMLSpy2005\Examples\ipo.xsd
C:\Program Files\Altova\XMLSpy2005\Examples\NanonullOrg.xsd
C:\Program Files\Altova\XMLSpy2005\Examples\OrgChart.xsd

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.


SchemaAgent Extended Validation for C:\Program Files\Altova\XMLSpy2005\Examples\address.xsd

Schemas to validate:

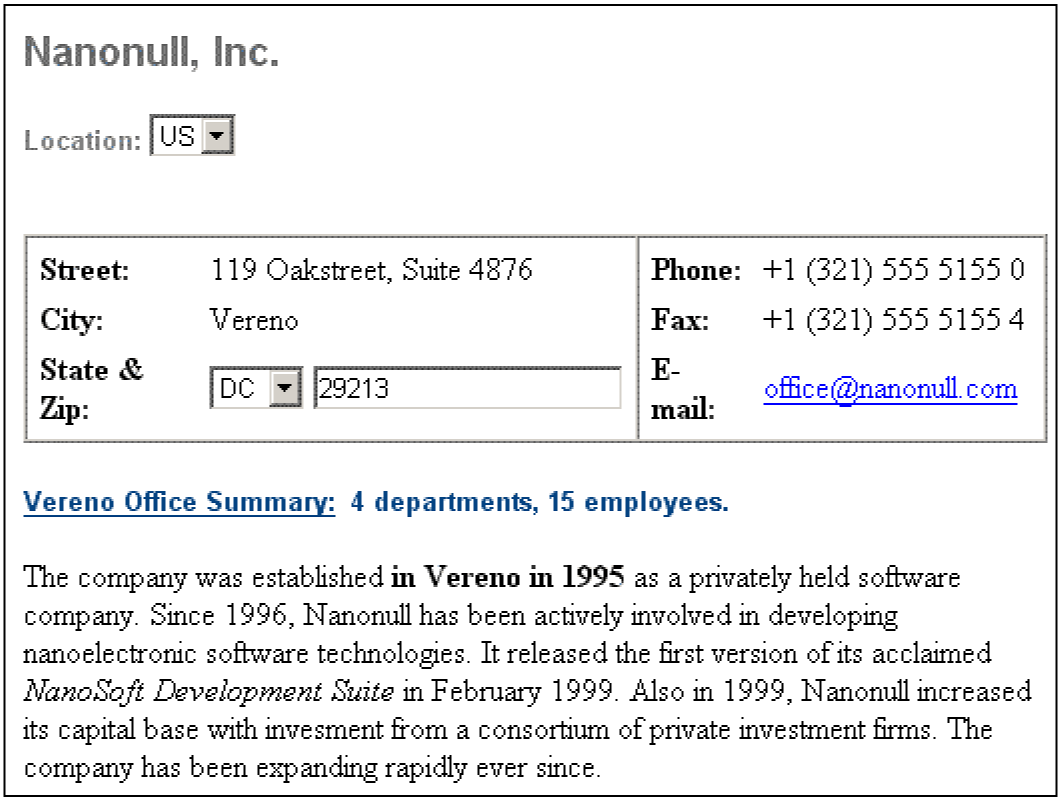
Schema files	Validation
C:\Program Files\Altova\XMLSpy2005\Examples\address.xsd	OK
C:\Program Files\Altova\XMLSpy2005\Examples\ipo.xsd	OK
C:\Program Files\Altova\XMLSpy2005\Examples\NanonullOrg.xsd	OK
C:\Program Files\Altova\XMLSpy2005\Examples\OrgChart.xsd	Failed

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 Authentic View

Authentic View (*see screenshot*) 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: 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 invesment from a consortium of private investment firms. The company has been expanding rapidly ever since.

Authentic View enables you to edit XML documents and databases (DBs) in an easy-to-use graphical user interface (GUI).

To be able to view and edit an XML document in Authentic View, the XML document must be associated with a **StyleVision Power Stylesheet,** which is created in Altova's StyleVision product. A StyleVision Power Stylesheet (`.sps` file) is, in essence, an XSLT stylesheet. It specifies an output presentation for an XML file that also includes data-entry mechanisms. Authentic View users can, therefore, write data back to the XML file or DB. A StyleVision Power Stylesheet is based on a schema and is specific to it. If you wish to use a StyleVision Power Stylesheet to edit an XML file in Authentic View, you must use a StyleVision Power Stylesheet 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 Document Window. If a StyleVision Power Stylesheet is not already assigned to the XML file, you will be prompted to assign one to it. You must use a StyleVision Power Stylesheet 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 StyleVision Power Stylesheet 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 StyleVision Power Stylesheet. After the Authentic View of an XML file is displayed, you can enter data in it and save the file.

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
- A detailed description of how to use various Authentic View features

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.

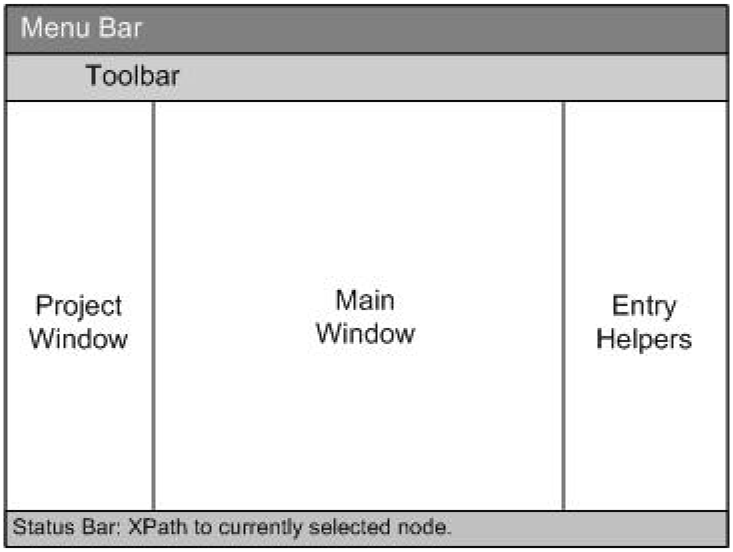
6.1 Interface

This section describes the Authentic View user interface. It contains the following sections:

- Overview of the GUI
- Authentic View toolbar icons
- Authentic View main window
- Authentic View entry helpers
- Authentic View context menus

6.1.1 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, HTML 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.

6.1.2 Authentic View toolbar icons

Icons in the Authentic View toolbar are command shortcuts. Some icons will already be familiar to you from other Windows applications or Altova products, others might be new to you. This section describes icons unique to Authentic View.

In the description below, related icons are grouped together.

Switching to Authentic View

If the XML document **is linked** to a StyleVision Power Stylesheet, **View | Authentic view** switches to Authentic View from another view.
If the document **is not linked** to a StyleVision Power Stylesheet, a dialog is displayed that asks you to link the document to a StyleVision Power Stylesheet. 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.

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.

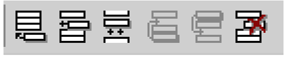
Show mixed markup. In the StyleVision Power Stylesheet, each XML element or attribute can be specified to display (as either large or small markup), or not display at all, in mixed markup mode. In mixed markup mode, therefore, the Authentic View user sees a customized markup. Note, however, that this customization is created by the person who has designed the StyleVision Power Stylesheet.

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

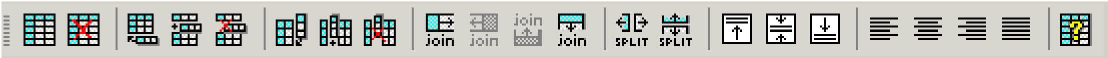
Move current row down by one row

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..

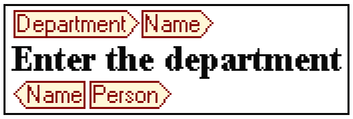
This icon opens the Edit Database Query dialog in which you can enter a query. Authentic View displays the queried record/s.

6.1.3 Authentic View main window

There are four viewing modes in Authentic View: Large Markup; Small Markup; Mixed Markup; and Hide All Markup. These modes enable you to view the document with varying levels of markup information.

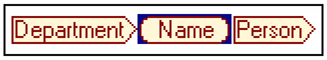
To switch between modes, use the commands in the **Authentic** menu or the icons in the toolbar (see the previous section, Authentic View toolbar icons).

Large markup
This shows the start and end tags of elements and attributes with the element/attribute names in the tags:



Department Name
Enter the department
Name Person

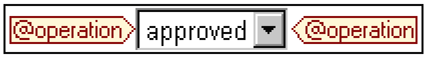
The element `Name` in the figure above is **expanded**, i.e. the start and end tags, as well as the content of the element, are shown. An element/attribute can be **contracted** by double-clicking either its start or end tag:



Department Name Person

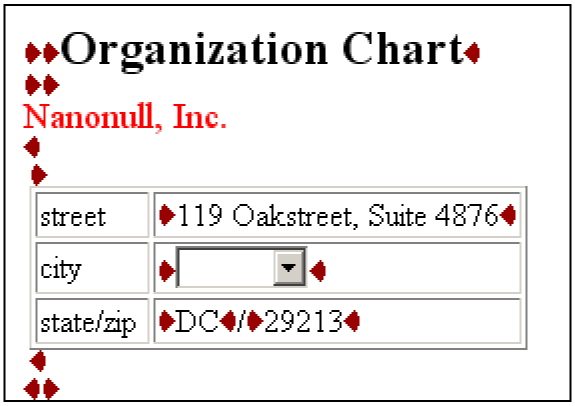
To expand the contracted element/attribute, double-click the contracted tag.

In large markup, attributes are recognized by the symbol `@` in the start and end tags of the attribute:



@operation approved @operation

Small markup
This shows the start and end tags of elements/attributes without names:

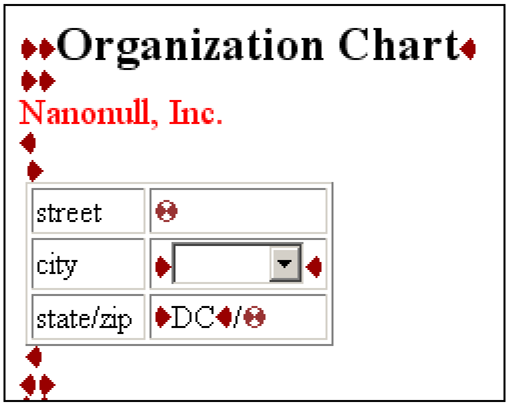


Organization Chart

Nanonull, Inc.

street	119 Oakstreet, Suite 4876
city	
state/zip	DC/29213

To contract and expand an element/attribute, double-click the appropriate tag. The example below shows two contracted elements in the table:



Organization Chart

Nanonull, Inc.

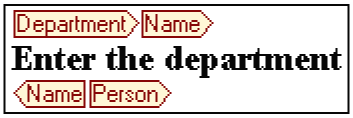
street	
city	
state/zip	DC/

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.

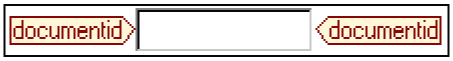
Content display
In Authentic View, content is displayed in two ways:

- Plain text. You type in the text, and this text becomes the content of the element or the value of the attribute.

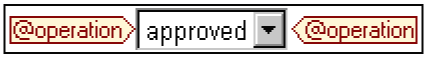


Enter the department

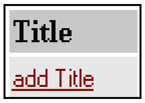
- Data-entry devices. The display contains either an input field (text box), a multiline input field, combo box, check box, or radio button. In the case of input fields and multiline input fields, the text you enter in the field becomes the XML content of the element or the value of the attribute.



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 the selection "approved" in the display example below could map to an XML value of "1", or to "approved", or anything else; while "not approved" in the display could map to "0", or "not approved", or anything else.



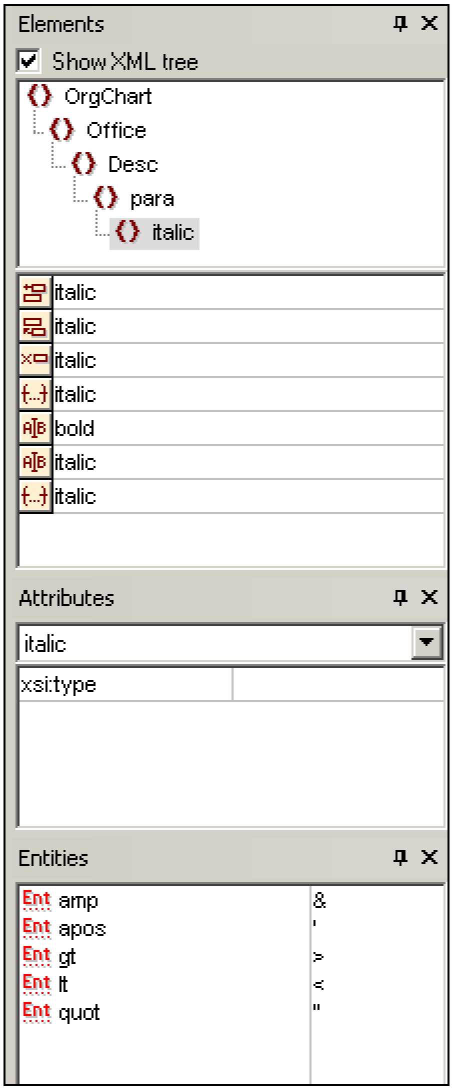
Optional nodes
When an element or attribute is **optional** (according to the referenced schema), a prompt of type "add [*element/attribute*]" is displayed:



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.

6.1.4 Authentic View entry helpers

There are three entry helpers in Authentic View: for Elements, Attributes, and Entities. They are displayed as windows down the right side of the Authentic View interface.



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 elements that can be appended after, inserted before, inserted within, applied to (i.e. replace), 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 an element from the Entry Helper, click its icon.

Append After Element

The element in the Entry Helper is appended 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), this icon indicates that the element in the Entry Helper can be applied to the selected (original) element. The applied element **replaces** the original 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.

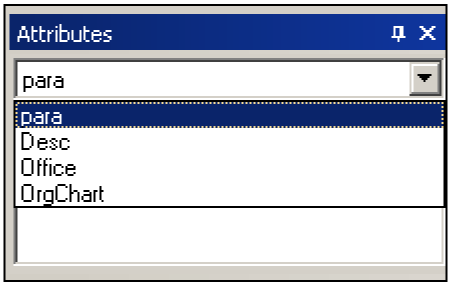
Clear Element (when range selected)
This icon appears to the left of the inline element within which the selected text range is. Clicking the icon clears that inline element but not the selected text range.

Clear Element (when insertion point selected)
This icon appears to the left of the inline element within which the cursor is. Clicking the icon clears that inline element but not its contents.

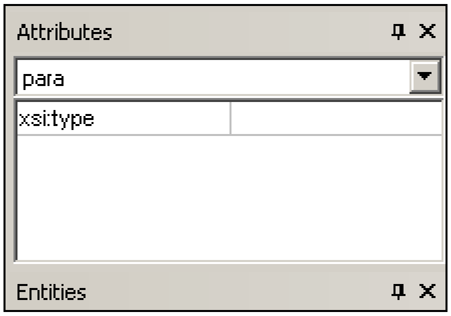
Attributes Entry Helper

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.)

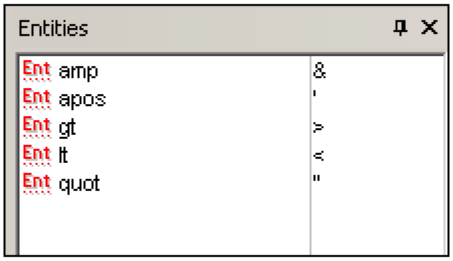


To enter a value for an attribute, click in the value field of the attribute and enter the value. This creates the attribute and its value in the XML document.

Please note: Entering a value for an attribute will have an effect in the Authentic View display only if such an effect has been specified in the StyleVision Power Stylesheet.

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.



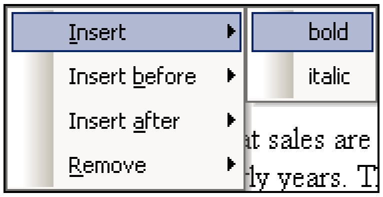
Ent amp	&
Ent apos	'
Ent gt	>
Ent lt	<
Ent quot	"

Please 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.

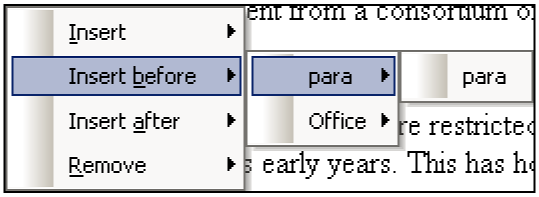
You can also **define your own entities** in Authentic View: see Define Entities in the How To Use section.

6.1.5 Authentic View context menus

Right-clicking on some selected document content or node pops up a context menu with commands relevant to the selection or cursor location. The context menu is shown below. 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.



Most of the commands available in the context menu are explained in Authentic View entry helpers.

Remove element

Positioning the mouse cursor over the **Remove** command pops up a menu list consisting of the selected element and all its ancestors up to the document element. Click the element to be removed. This is a quick way to delete an element or any of its ancestors. Note that clicking an ancestor element will remove all its descendants, including the selected element.

6.2 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 commonly used or require an explanation of the mechanisms or concepts involved.

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.
- What image formats can be displayed in Authentic View.

6.2.1 Tables in Authentic View

The three table types fall into two categories: SPS tables (static and dynamic) and XML tables.

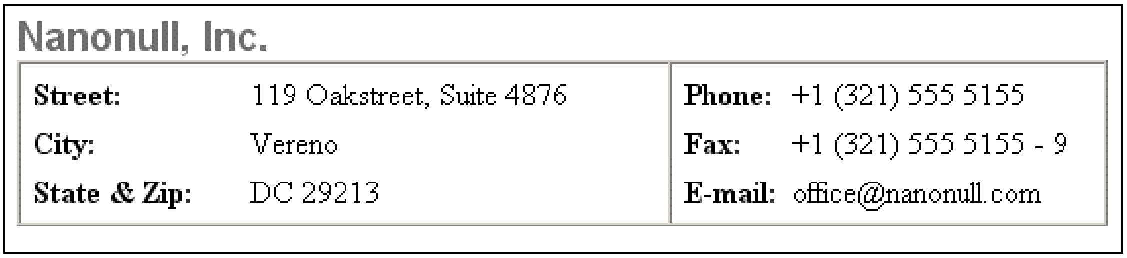
SPS tables are of two types: static and dynamic. SPS tables are designed by the designer of the StyleVision Power Stylesheet to which your XML document is linked. You yourself cannot insert an SPS table into the XML document, but you can enter data into SPS table fields and add and delete the rows of dynamic SPS tables. The section on SPS tables below explains the features of these tables.

XML 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 XML tables and the XML 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.



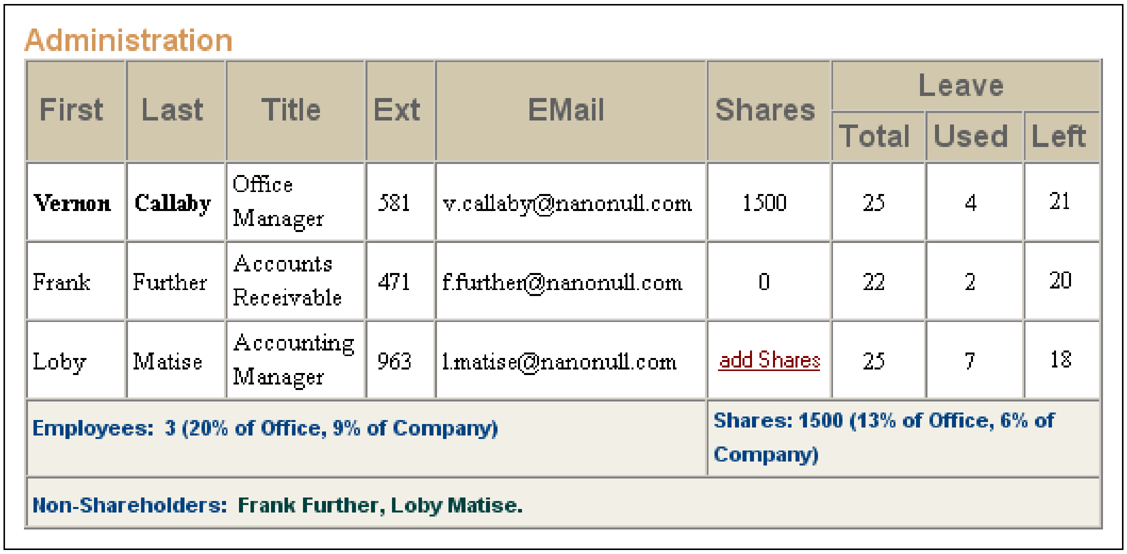
Nanonull, Inc.

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.

XML Tables

XML tables can be inserted by you, the user of Authentic View. They enable you to insert tables anywhere in the XML document where they are allowed, which is useful if you need to insert tabular information in your document. These tables will be printed out as tables when you print out directly from Authentic View. If you are also generating output with XSLT stylesheets, discuss the required output with the designer of the StyleVision Power Stylesheet.

Note that you can insert XML tables only at allowed locations. These locations are specified in the schema (DTD or XML Schema). If you wish to insert a table at additional locations, discuss this with the person designing the StyleVision Power Stylesheet.

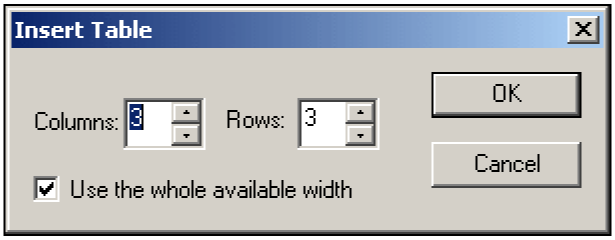
Working with XML tables

There are three steps involved when working with XML tables: inserting the table; formatting it; and entering data. The commands for working with XML tables are available as icons in the toolbar (see XML table editing icons).

Inserting tables

To insert an XML table:

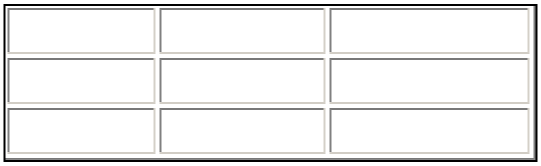
1. Place your cursor where you wish to insert the table, and click the ▦ icon. (Note that where you can insert tables is determined by the schema.) This opens the Insert Table dialog (shown below).



Columns: 3 Rows: 3

☑ Use the whole available width

2. Select the number of columns and rows, and specify whether you wish the table to extend the entire available width. For the specifications given in the dialog box shown above, the following table is created.

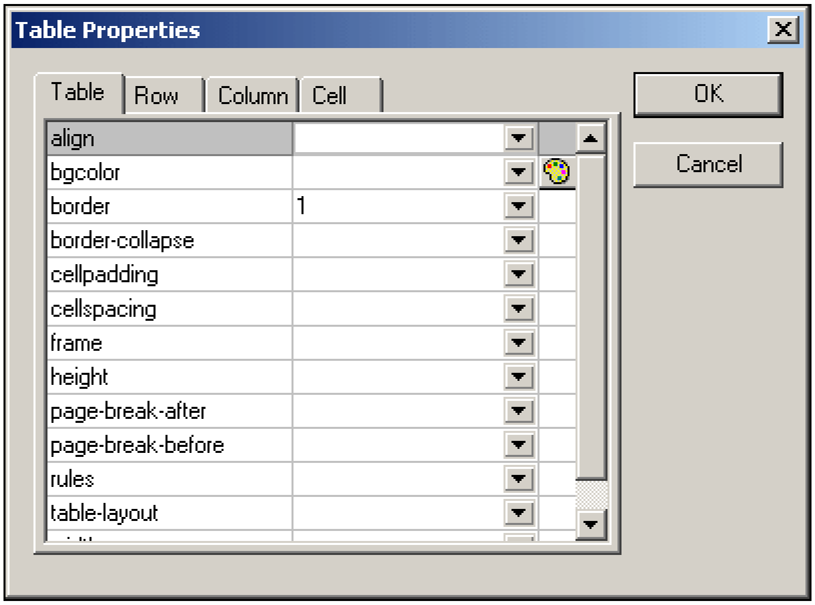


You can add and delete columns, create row and column joins later. Create the broad structure first.

Please note: All modifications to table structure must be made by using the **Table** menu commands. They cannot be made by changing attribute values in the Attribute Entry Helper.

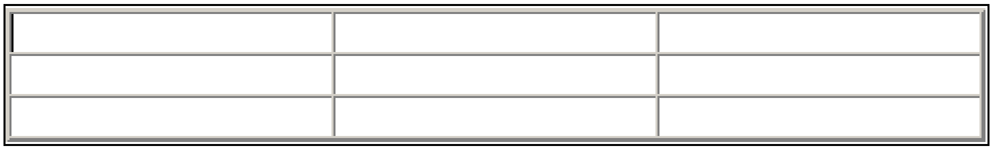
Formatting tables and entering data
To format your table:

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.



align	
bgcolor	
border	1
border-collapse	
cellpadding	
cellspacing	
frame	
height	
page-break-after	
page-break-before	
rules	
table-layout	

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.

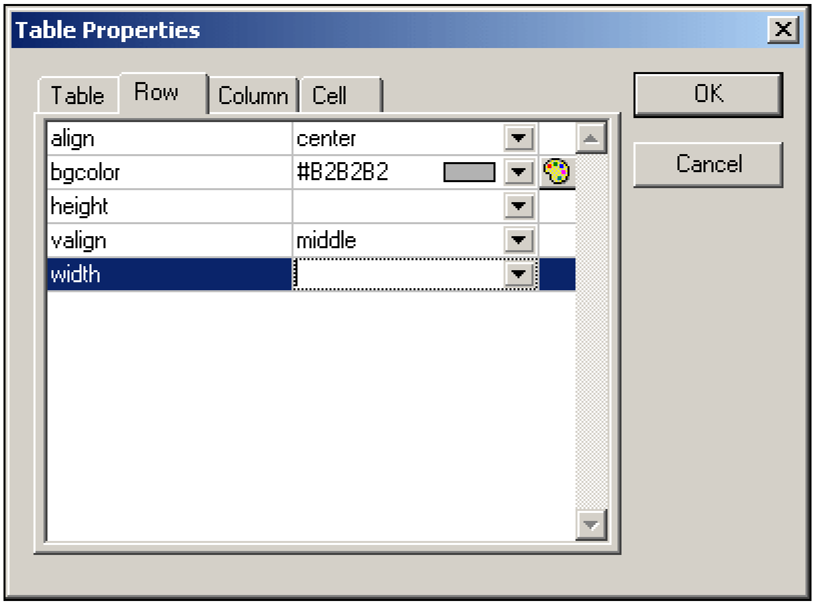
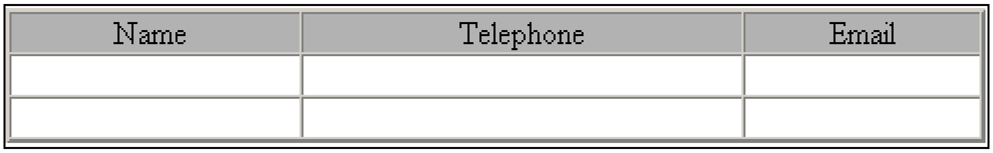


Table Properties

align	center
bgcolor	#B2B2B2
height	
valign	middle
width	

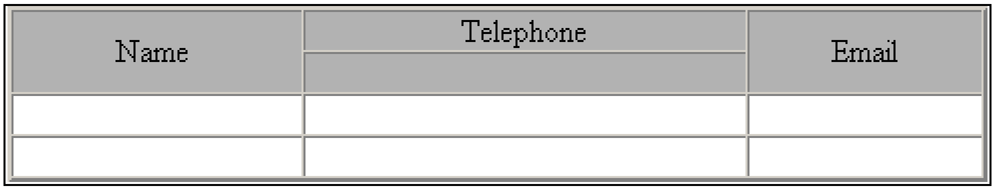
Since the first row will be the header row, set a background color to differentiate this row from the other rows. Note the Row properties that have been set in the figure above. Then enter the column header text. Your table will now look like this:



Name	Telephone	Email

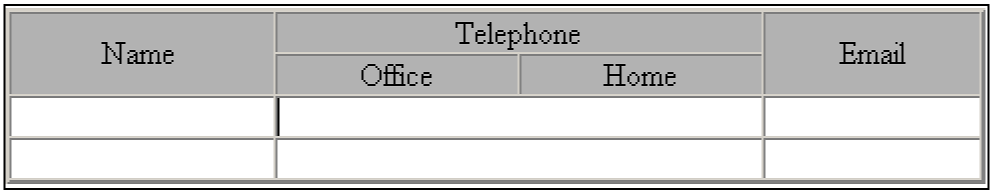
Notice that the alignment is centered as specified.

4. Now, say you want to divide the "Telephone" column into the sub-columns "Office" and "Home", in which case you would need to join cells. Place the cursor in the "Telephone" cell, and click the (Split vertically) icon. Your table will look like this:



Name	Telephone	Email

5. 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	

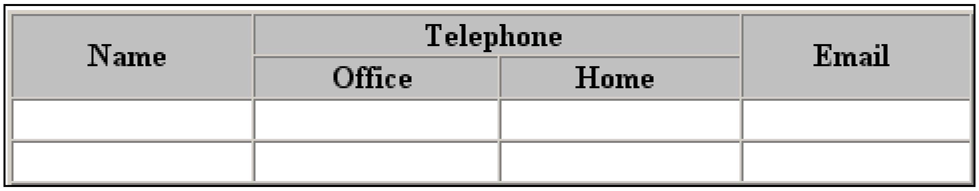
Now you will have to vertically split each cell in the "Telephone" column.

You can also add and delete columns and rows, and vertically align cell content, using the table-editing icons. The XML table editing icons are described in the User Reference, in the section titled "XML Table Icons".

Moving among cells in the table
To move among cells in the XML 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 an XML 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.

XML Table Editing Icons

The commands required to edit XML 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 XML tables are to be used, see XML 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.

Join cell right

The "Join cell right" command joins the current cell (current cursor position) with the cell to the right. The tags of both cells remain in the new cell, the column headers remain unchanged.

Join cell below

The "Join cell below" command joins the current cell (current cursor position) with the cell below. The tags of both cells remain in the new cell, the column headers remain unchanged.

Join cell above

The "Join cell above" command joins the current cell (current cursor position) with the cell above. The tags of both cells remain in the new cell, the column headers remain unchanged.

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.

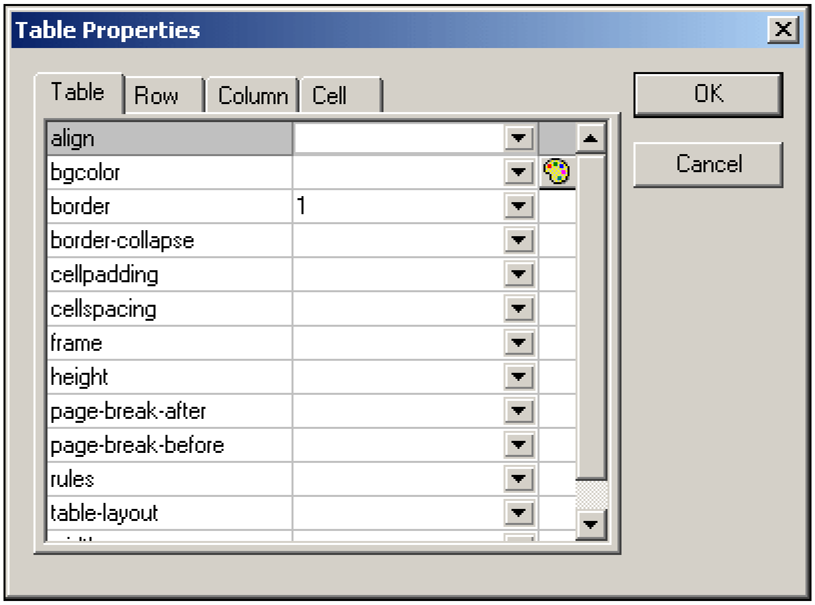


Table Properties

align	
bgcolor	
border	1
border-collapse	
cellpadding	
cellspacing	
frame	
height	
page-break-after	
page-break-before	
rules	
table-layout	

6.2.2 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:

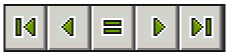
- The number of records in a DB table that are displayed in Authentic View may have been deliberately restricted by the designer of the StyleVision Power Stylesheet in order to make the design more compact. In such cases, only that limited number of records is initially loaded into Authentic View. Using 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:

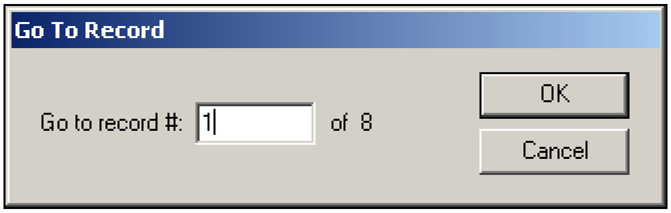
- Click **Authentic | Edit Database Data**, and browse for the required StyleVision Power Stylesheet.

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; Go to Previous Record; Open the Go to Record dialog (*see screenshot*); Go to Next Record; and Go to Last Record.



Go To Record

Go to record #: 1 of 8

To navigate a DB table, click the required button.

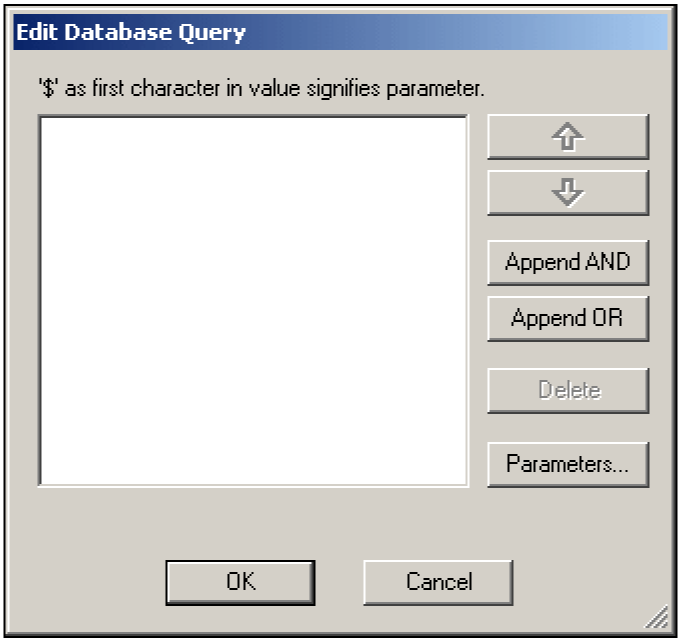
DB Queries

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.

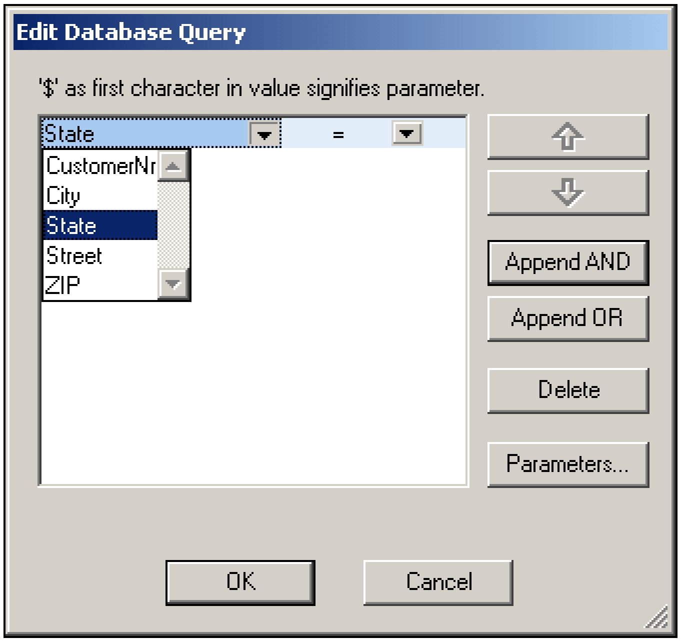
Please note: If you get a message saying that too many tables are open, then you can reduce the number of tables that are open by using a query to filter out some tables.

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

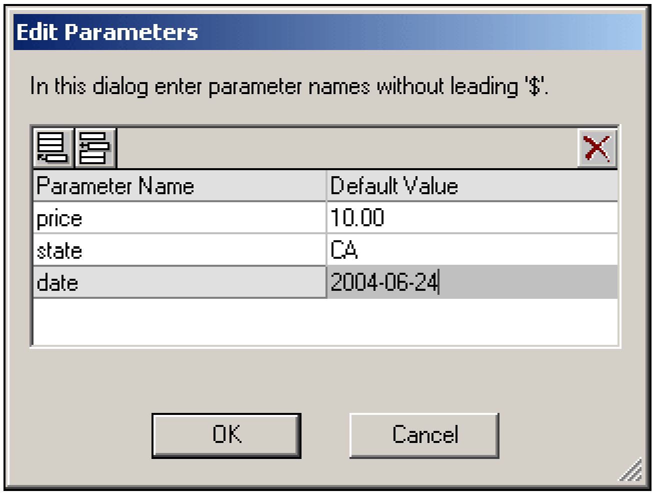
You can enter the name of a **parameter** as the value of an expression when creating queries. Parameters are variables that can be used instead of literal values in queries. You first declare the parameter and its value, and then use the parameter in expressions. This causes the value of the parameter to be used as the value of that expression. The parameters that you add in the Edit Parameters dialog can be parameters that have already been declared for the stylesheet. In this case, the new value overrides the value in the stylesheet.

Parameters are useful if you wish to use a single value in multiple expressions.

Declaring parameters from the Edit DB Query dialog

To declare parameters:

1. Click the **Parameters...** button in the Edit Database Query dialog. This opens the **Edit Parameters** dialog (*see screenshot*).



In this dialog enter parameter names without leading '$'.

Parameter Name	Default Value
price	10.00
state	CA
date	2004-06-24

2. Click Append or Insert .
3. Type in the name and value of the parameter in the appropriate fields.

Please note: The Edit Parameters dialog contains **all** the parameters that have been defined for the stylesheet. While it is an error to use an undeclared parameter in the StyleVision Power Stylesheet, it is not an error to declare a parameter and not use it.

Using parameters in queries

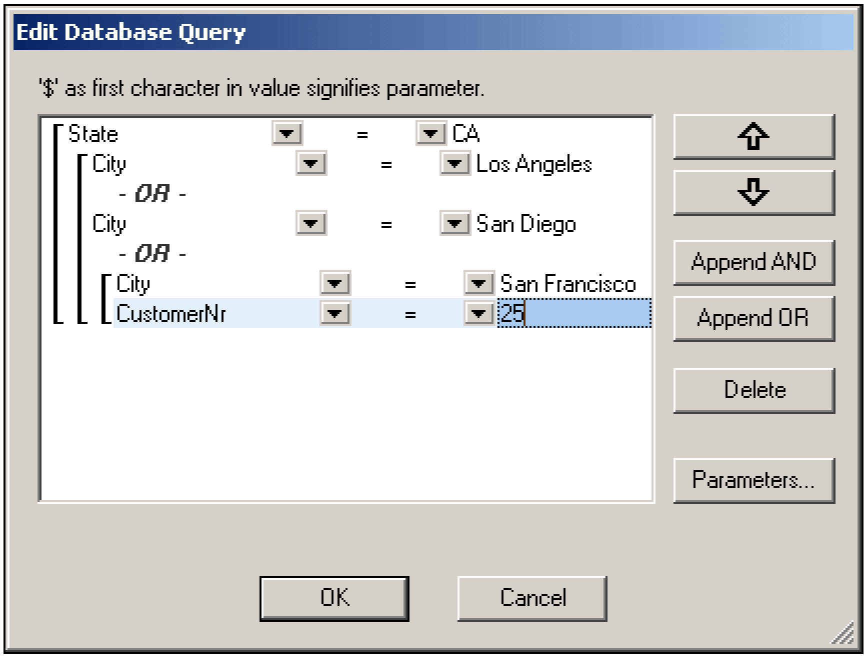
To enter the name of a parameter as the value of an expression:

- Type $ into the value input field followed (without any intervening space) by the name of the parameter in the Edit Database Query dialog.

Please note: If the parameter has already been declared, then the entry will be colored green. If the parameter has not been declared, the entry will be red, and you must declare it.

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.



'$' as first character in value signifies parameter.

State = CA
City = Los Angeles
- OR -
City = San Diego
- OR -
City = San Francisco
CustomerNr = 25

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 StyleVision 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 (using the navigation icons described above).

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.

6.2.3 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.

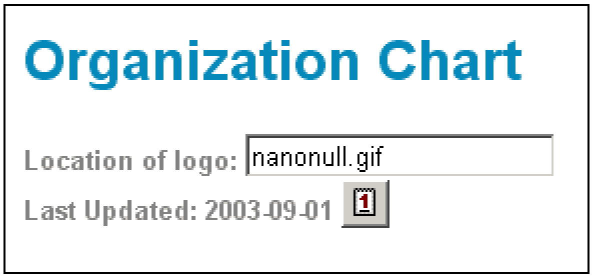
Note on date formats

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*).

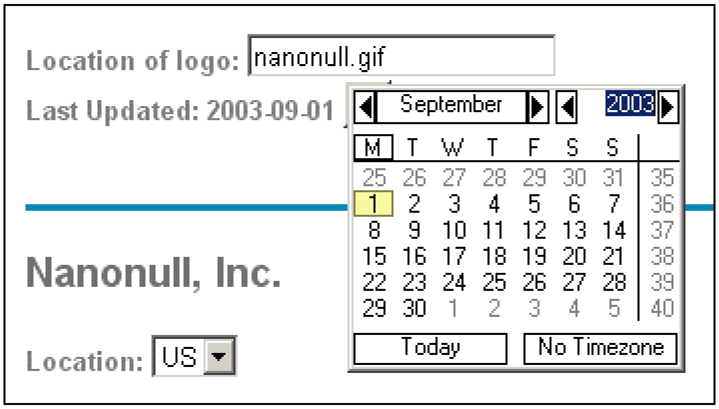


Organization Chart

Location of logo: nanonull.gif

Last Updated: 2003-09-01

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

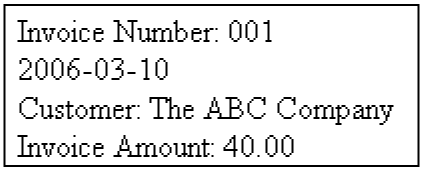
M	T	W	T	F	S	S	
25	26	27	28	29	30	31	35
1	2	3	4	5	6	7	36
8	9	10	11	12	13	14	37
15	16	17	18	19	20	21	38
22	23	24	25	26	27	28	39
29	30	1	2	3	4	5	40

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

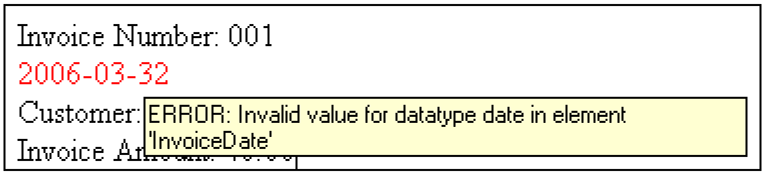
For date fields that do not have a Date Picker (*see screenshot*), you can edit the date directly by typing in the new value.

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

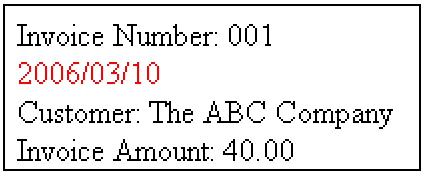
If you edit a date and change it such that it is out of the valid range for dates, the date turns red to alert you to the error. If you place the mouse cursor over the invalid date, an error message appears (*see screenshot*).



Invoice Number: 001
2006-03-32
Customer:
Invoice A

ERROR: Invalid value for datatype date in element 'InvoiceDate'

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

6.2.4 Defining Entities

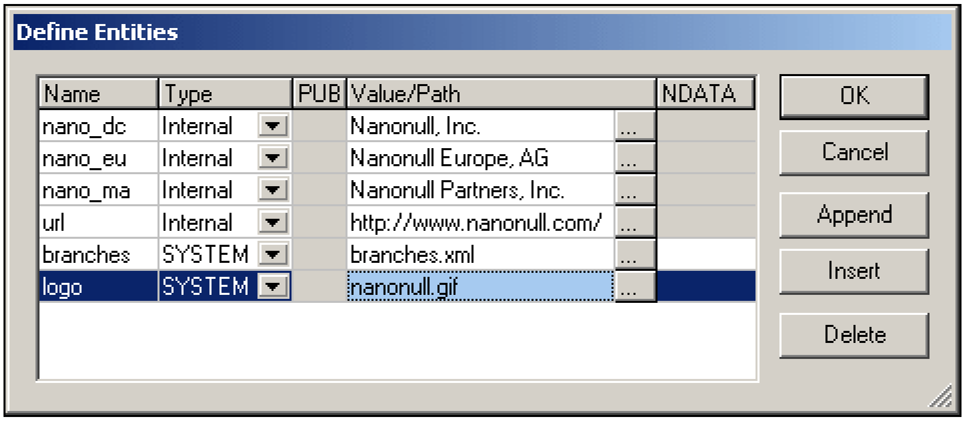
You can define entities for use in Authentic View, whether your document is based on a DTD or an XML Schema. Once defined, these entities are displayed in the Entities Entry Helper and in the **Insert Entity** submenu of the context menu. When you double-click on an entity in the Entities Entry Helper, 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.



Define Entities

Name	Type	PUB	Value/Path	NDATA
nano_dc	Internal		Nanonull, Inc.	
nano_eu	Internal		Nanonull Europe, AG	
nano_ma	Internal		Nanonull Partners, Inc.	
url	Internal		http://www.nanonull.com/	
branches	SYSTEM		branches.xml	
logo	SYSTEM		nanonull.gif	

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. Note that entities are a good mechanism for including Unicode characters in your document; do this by entering the Unicode number as the value of an internal entity.
   - 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 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.

Limitations of entities

- An entity contained within another entity is not resolved, either in the dialog, Authentic View, or XSLT output, and the ampersand character of such an entity is displayed in its escaped form, i.e. `&amp;`.
- External 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 of type `ENTITY` or `ENTITIES`. Such entities are resolved when the document is processed with an XSLT generated from the StyleVision Power Stylesheet.

6.2.5 Images in Authentic View

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)

6.2.6 Keystrokes in Authentic View

Enter (Carriage Return) Key
In Authentic View the **Return** 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 **Return** 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 chapters, 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 **Return** key does **not** insert a carriage return/line feed, i.e. it does not jump to a new line. This is the case even when the cursor is inside a text node, such as paragraph.

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 the browser.
- HTML files which are either created directly as HTML or created via an XSLT transformation of an XML file.

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 XSL in IE 5 is not 100% compatible with the official XSLT Recommendation. So if you encounter problems, with Browser View with IE 5, you should upgrade to IE 6.)

To view XML and HTML files in Browser View, click the **Browser** tab.

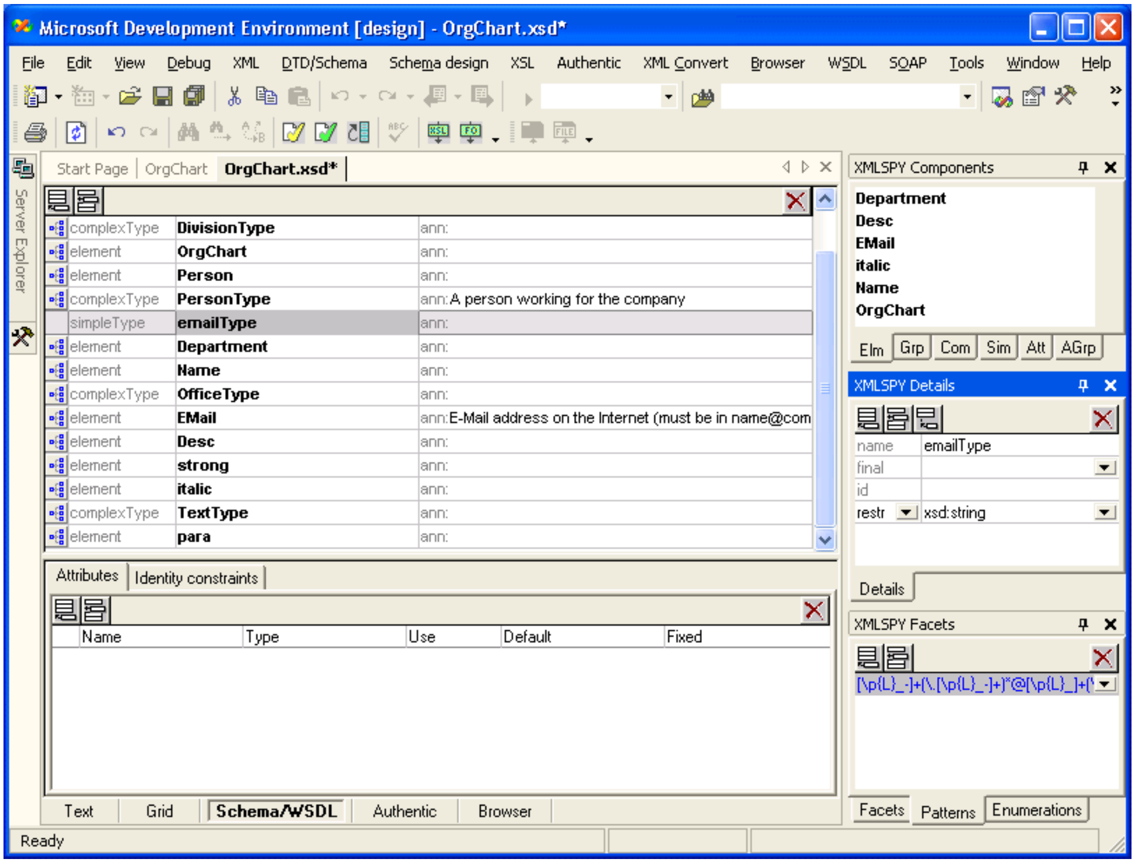
Browser View features

- You can open the Browser View in a separate window. To do this, switch to Browser View, and select the menu command **Browser | Separate window**. This allows you to tile windows so that you see the Browser View side-by-side with an editing view. As a result, any change you make in the editing view can be seen immediately in the Browser View: Simply press **F5** in the editing view or make the Browser View window the active window (by clicking on it).
- Browser View supports Find. In Browser View, select the menu command **Edit | Find** to find text strings.
- Browser View supports common browser commands: Back, Forward, Stop, Refresh, Font Size, and Print.

8 XMLSpy in MS Visual Studio .NET

You can integrate your version of XMLSpy into the Microsoft Visual Studio .NET IDE versions 2002, 2003 and 2005. This unifies the best of both worlds, integrating advanced XML editing capabilities with the advanced development environment of Visual Studio .NET. To do this, you need to do the following:

- Install Microsoft Visual Studio .NET
- Install XMLSpy (Enterprise or Professional Edition)
- Download and run the XMLSpy integration for Microsoft Visual Studio .NET package. 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 edition (Enterprise or Professional).)


Microsoft Development Environment [design] - OrgChart.xsd*
File Edit View Debug XML DTD/Schema Schema design XSL Authentic XML Convert Browser WSDL SOAP Tools Window Help
Server Explorer
Start Page
OrgChart
OrgChart.xsd*
XMLSPY Components
Department
Desc
EMail
italic
Name
OrgChart
Elm Grp Com Sim Att AGrp
XMLSPY Details
Details
XMLSPY Facets
Facets Patterns Enumerations
Attributes Identity constraints
Name Type Use Default Fixed
Text Grid Schema/WSDL Authentic Browser
Ready

complexType	DivisionType	ann:
element	OrgChart	ann:
element	Person	ann:
complexType	PersonType	ann:A person working for the company
simpleType	emailType	ann:
element	Department	ann:
element	Name	ann:
complexType	OfficeType	ann:
element	EMail	ann:E-Mail address on the Internet (must be in name@com
element	Desc	ann:
element	strong	ann:
element	italic	ann:
complexType	TextType	ann:
element	para	ann:

name	emailType
final	
id	
restr	xsd:string

[\p{L}_-]+(\.[\p{L}_-]+)*@[\p{L}_]+('

Once the integration package has been installed, you will be able to use XMLSpy in the Visual Studio .NET environment.

8.1 Differences between .NET and standalone versions

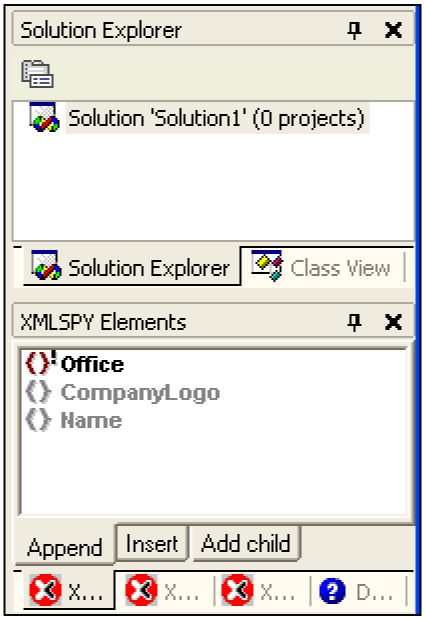
The Enterprise and Professional XMLSpy Visual Studio .NET Edition have the same functionality in both Visual Studio .Net 2002 and 2003.

XMLSpy Menus or menu options visible on Visual Studio .NET startup:

- **File | New | AUTHENTIC File**
- **XML Convert**
- **SOAP**
- **Tools | XMLSPY Options**

Entry helpers (Tool Windows in Visual Studio .NET)

- Entry helper windows are grouped in one tab group below the Solution Explorer window on startup.
- You can drag the 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 autohide.



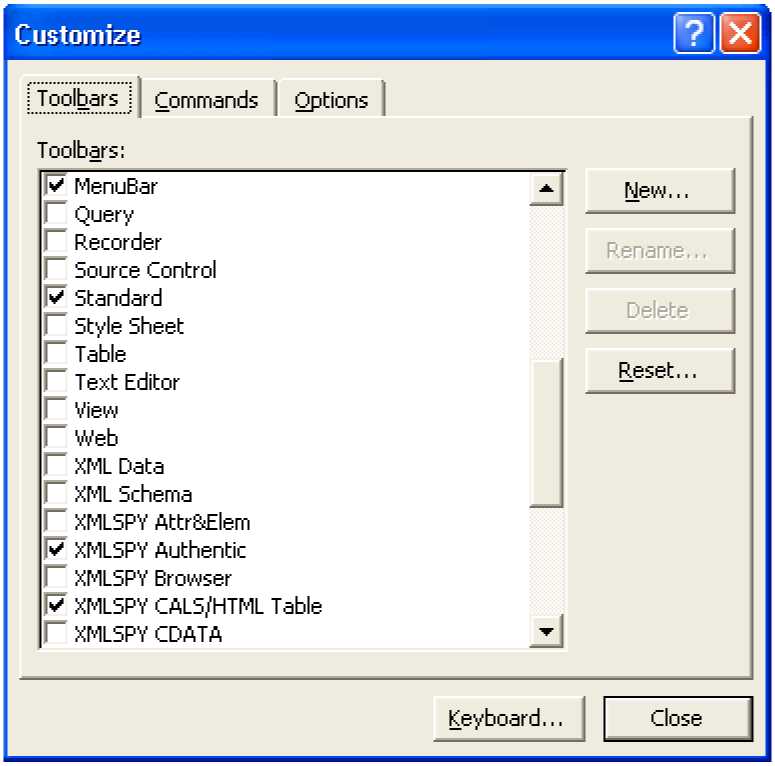
Changed functionality in the Visual Studio .NET editions:

Menu **File**
Use **File | Open | File from web** instead of **File | Open URL** to open a file from an URL.
Use **File | Save "my file" as | My network places** instead of **File | Save URL** to save a file to an URL.

Menu **Edit,** Undo and Redo
The Undo and Redo commands affect all actions (copy, paste, etc.) made in the development environment, including all actions in XMLSpy.

Please note: There are two sets of undo/redo icons. The XMLSpy icons are in the XMLSpy Main icon bar. The Visual Studio .NET icons are disabled while editing files opened with XMLSpy.

Menu **Tools | Customize | Toolbar**, **Commands** and **Options**.

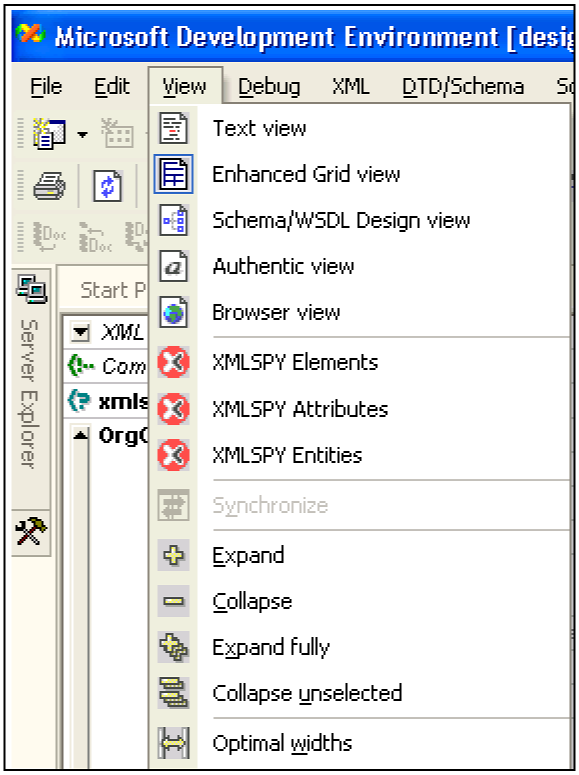


Toolbars:

☑ MenuBar
☐ Query
☐ Recorder
☐ Source Control
☑ Standard
☐ Style Sheet
☐ Table
☐ Text Editor
☐ View
☐ Web
☐ XML Data
☐ XML Schema
☐ XMLSPY Attr&Elem
☑ XMLSPY Authentic
☐ XMLSPY Browser
☑ XMLSPY CALS/HTML Table
☐ XMLSPY CDATA

These tabs contain both Visual Studio .NET and XMLSpy commands.

Menu **View**



The **View** menu allows you to select:

- the type of view you want to see your XML document in: Text, Enhanced Grid, Schema/WSDL, Authentic or Browser view.
- XMLSpy entry helper windows that may have been closed.
- XSLT and SOAP debugger Tool windows that may have been closed while a debugger is running.
- Grid view functions which make viewing data easier.

Menu **Help**
The **Help** menu contains the submenu XMLSpy Help, which is where you can open the XMLSpy help. It also contains links to the Altova Support center, Component download area, etc.

Unsupported features of the .NET edition of XMLSpy

Info window
The Info window is not supported. This window gives extra information on currently selected elements/attributes etc. E.g. the name of the element or attribute, the datatype, enumerations and occurrence.

Project window and **Project** menu (as well as source control submenus).
The **Project** menu (and Source control submenus) are not available, as MS Visual Studio .NET has its own project and source control environment. This means that batch validation, and batch conversion are currently not supported.

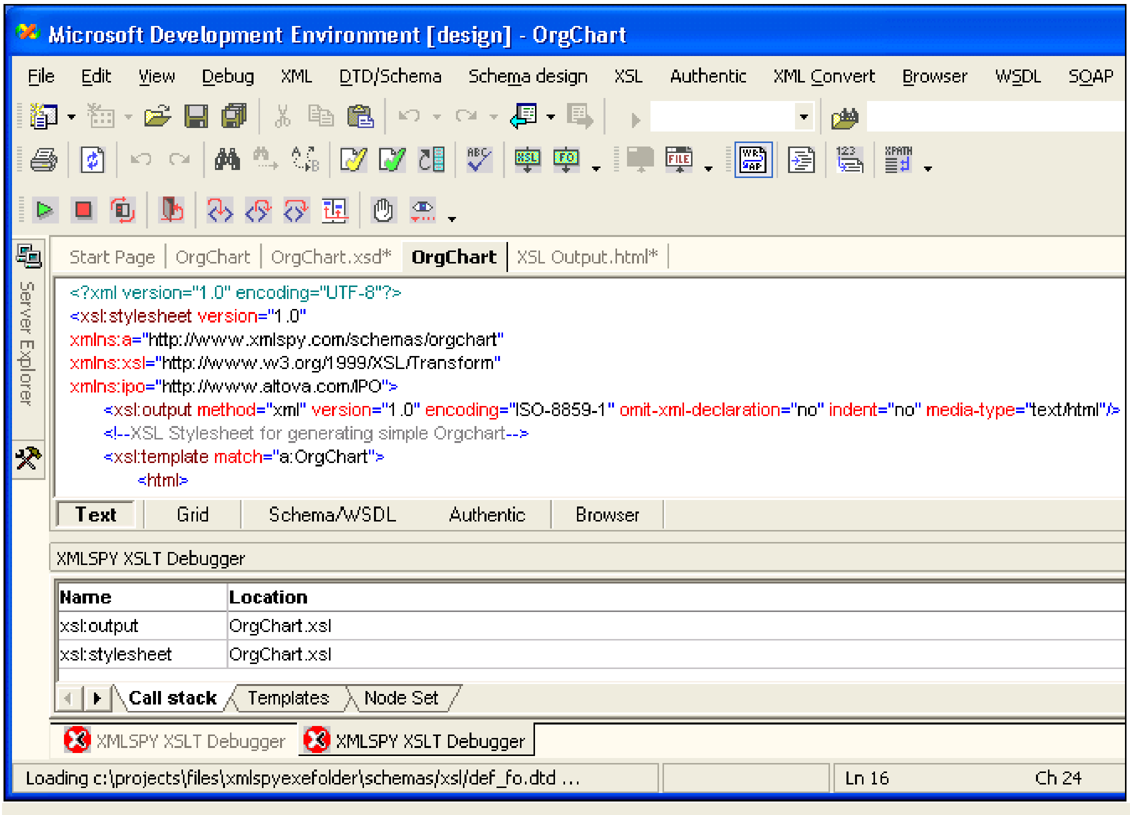
Tools | XML Spy Options | Scripting
The scripting environment is currently not available.

Separate Browser window
The "Show in separate window by default" check box in the menu **Tools | Options | View** tab, is not supported. This means that the Text and Browser view are always incorporated in the same window when you transform an XML file to HTML for example.

Authentic view
Text state icons are not available in the Authentic view. The functions or formatting that they may provide, are still available in the entry helper or context menu, however. Please see the Stylesheet Designer documentation for more information.

8.2 Visual Studio .NET and XSLT Debugger

The screenshot below displays how the `OrgChart.xml` file (found in the `Examples` folder) appears when the XSLT Debugger has been activated. By default, all files are displayed in one tab group. To make the debugging process easier to follow, you can create your own tab group in Visual Studio .NET.



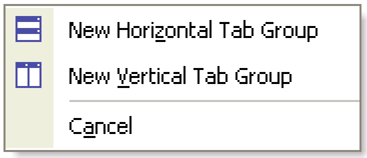
```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:a="http://www.xmlspy.com/schemas/orgchart"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:ipo="http://www.altova.com/IPO">
    <xsl:output method="xml" version="1.0" encoding="ISO-8859-1" omit-xml-declaration="no" indent="no" media-type="text/html"/>
    <!--XSL Stylesheet for generating simple Orgchart-->
    <xsl:template match="a:OrgChart">
        <html>
```

Name	Location
xsl:output	OrgChart.xsl
xsl:stylesheet	OrgChart.xsl

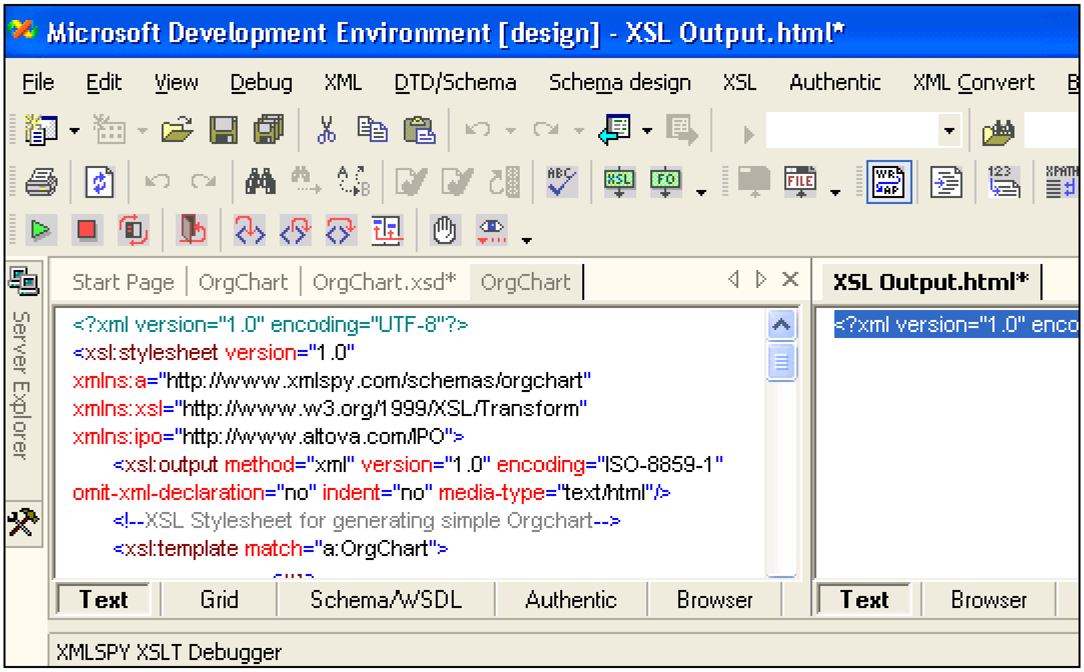
Loading c:\projects\files\xmlspyexefolder\schemas\xsl/def_fo.dtd ... Ln 16 Ch 24

To create a new tab group:

1. Click the **XSL Output.html** tab, then drag and drop it somewhere into the currently active tab. This opens a pop-up menu which allows you to define the type of tab you want to create.



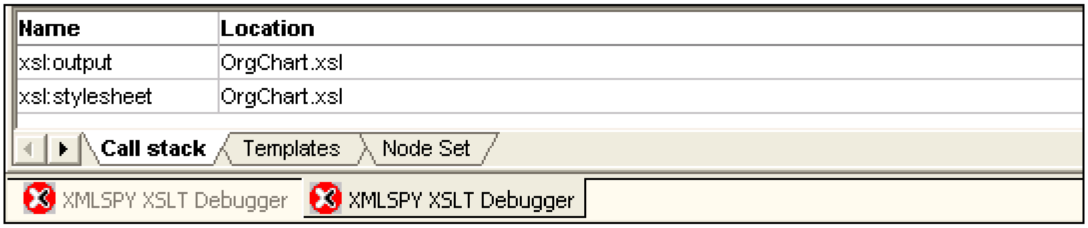
2. Select **New Vertical Tab Group**. This creates a new tab consisting of the XSL `Output.html` file.



```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:a="http://www.xmlspy.com/schemas/orgchart"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:ipo="http://www.altova.com/IPO">
    <xsl:output method="xml" version="1.0" encoding="ISO-8859-1"
omit-xml-declaration="no" indent="no" media-type="text/html"/>
    <!--XSL Stylesheet for generating simple Orgchart-->
    <xsl:template match="a:OrgChart">
```

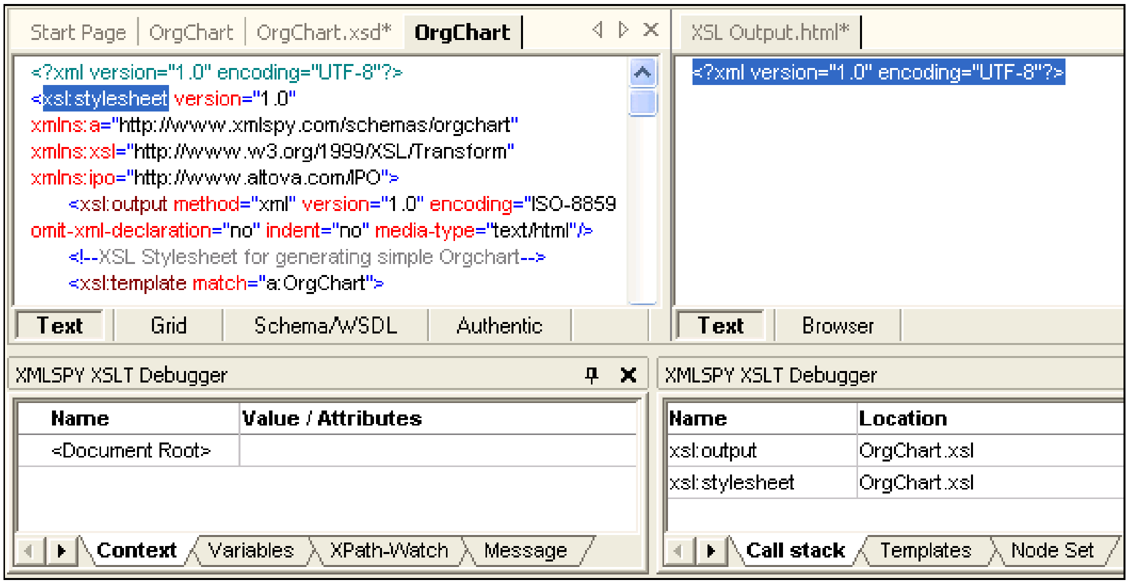
```
<?xml version="1.0" enco
```

3. Use the same method to create a new debug window tab group.



Name	Location
xsl:output	OrgChart.xsl
xsl:stylesheet	OrgChart.xsl

4. Drag the rightmost **XMLSPY XSLT Debugger** window to the right.
5. Select **New Vertical Tab Group** from the pop-up menu.



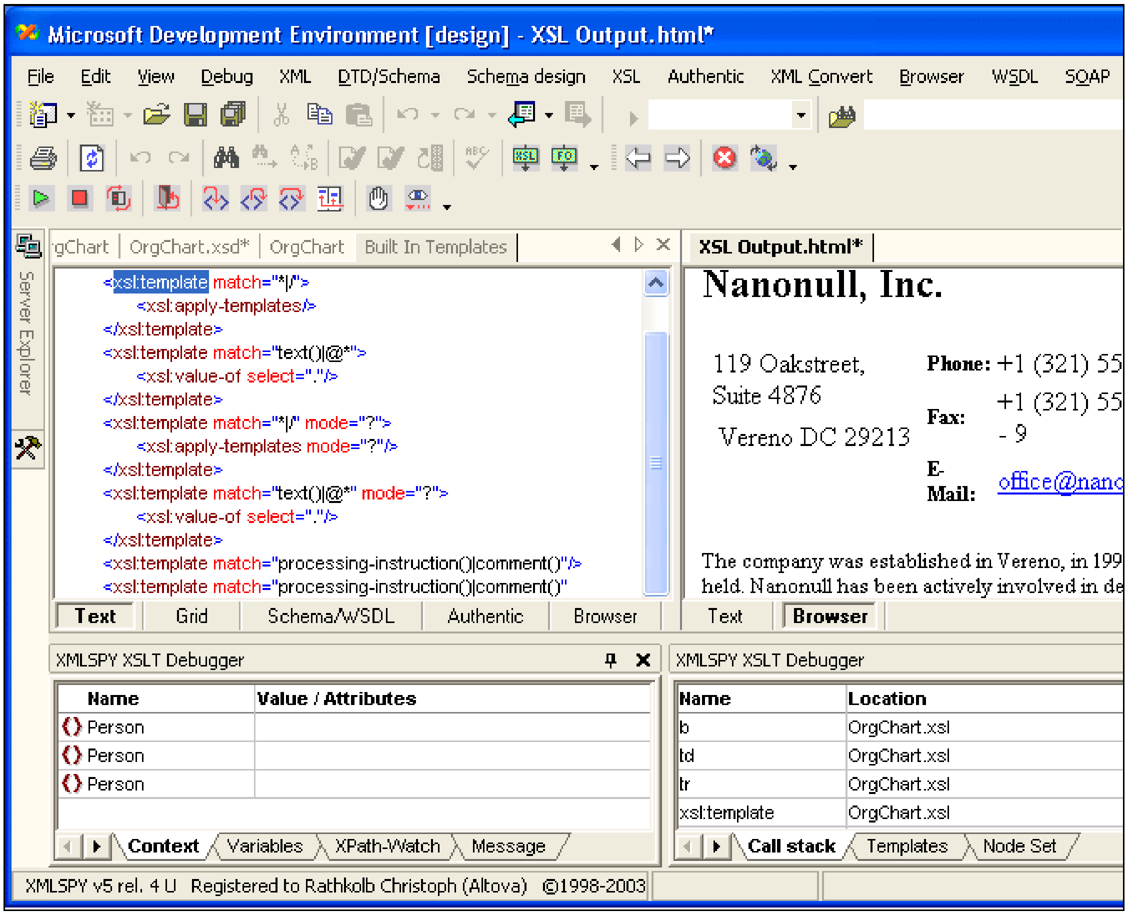
```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:a="http://www.xmlspy.com/schemas/orgchart"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:ipo="http://www.altova.com/IPO">
    <xsl:output method="xml" version="1.0" encoding="ISO-8859
omit-xml-declaration="no" indent="no" media-type="text/html"/>
    <!--XSL Stylesheet for generating simple Orgchart-->
    <xsl:template match="a:OrgChart">
```

```
<?xml version="1.0" encoding="UTF-8"?>
```

Name	Value / Attributes
<Document Root>	

Name	Location
xsl:output	OrgChart.xsl
xsl:stylesheet	OrgChart.xsl

View while debugging the `OrgChart.xml` file in XSLT Debugger:



Microsoft Development Environment [design] - XSL Output.html*
File Edit View Debug XML DTD/Schema Schema design XSL Authentic XML Convert Browser WSDL SOAP
Server Explorer
gChart OrgChart.xsd* OrgChart Built In Templates
XSL Output.html*
Text Grid Schema/WSDL Authentic Browser
Text Browser
XMLSPY XSLT Debugger
Context Variables XPath-Watch Message
Call stack Templates Node Set

```
<xsl:template match="*|/">
    <xsl:apply-templates/>
</xsl:template>
<xsl:template match="text()|@*">
    <xsl:value-of select="."/>
</xsl:template>
<xsl:template match="*|/" mode="?">
    <xsl:apply-templates mode="?"/>
</xsl:template>
<xsl:template match="text()|@*" mode="?">
    <xsl:value-of select="."/>
</xsl:template>
<xsl:template match="processing-instruction()|comment()"/>
<xsl:template match="processing-instruction()|comment()"
```

Nanonull, Inc.

119 Oakstreet,
Suite 4876
Vereno DC 29213

Phone: +1 (321) 55
Fax: +1 (321) 55
- 9
E-Mail: office@nanc

The company was established in Vereno, in 199
held. Nanonull has been actively involved in de

Name	Value / Attributes
Person	
Person	
Person	

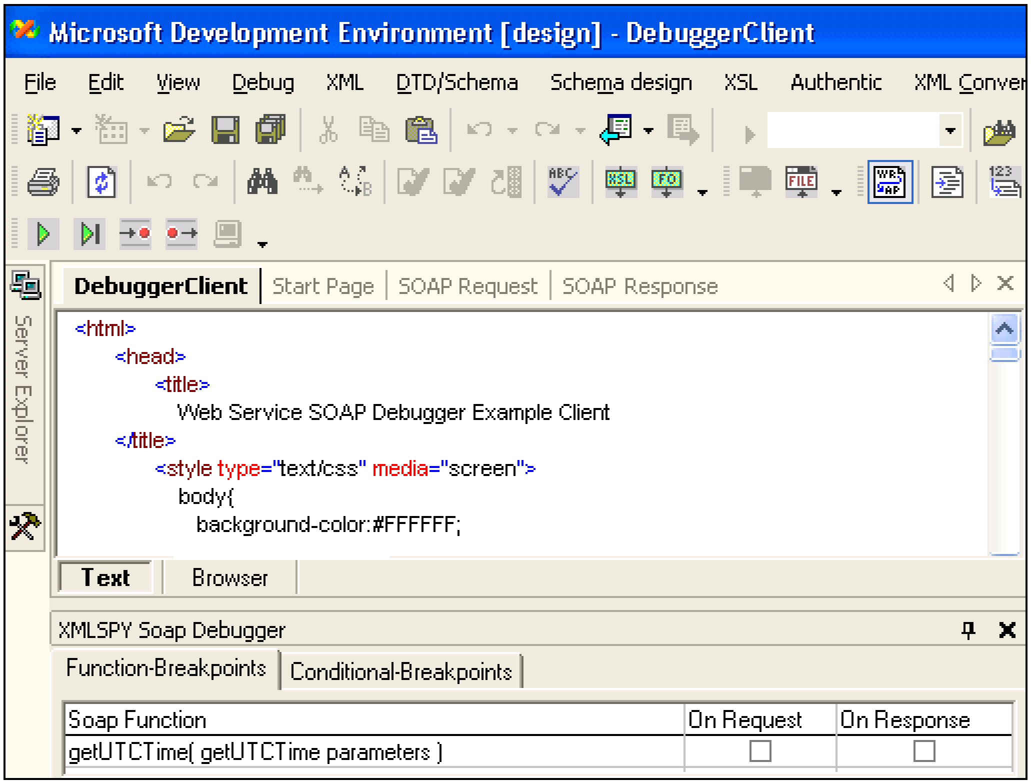
Name	Location
b	OrgChart.xsl
td	OrgChart.xsl
tr	OrgChart.xsl
xsl:template	OrgChart.xsl

XMLSPY v5 rel. 4 U Registered to Rathkolb Christoph (Altova) ©1998-2003

8.3 Visual Studio .NET and SOAP Debugger

SOAP Debugger

The screenshot below displays how the `DebuggerClient.html` file appears when the SOAP Debugger has been activated. Per default, all SOAP windows are displayed in one tab group. To make the debugging process easier to follow, you can create your own tab group in Visual Studio .NET.

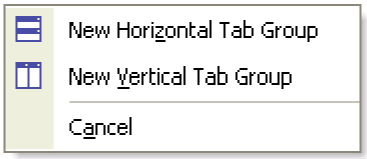


```
<html>
    <head>
        <title>
            Web Service SOAP Debugger Example Client
    </title>
        <style type="text/css" media="screen">
            body{
              background-color:#FFFFFF;
```

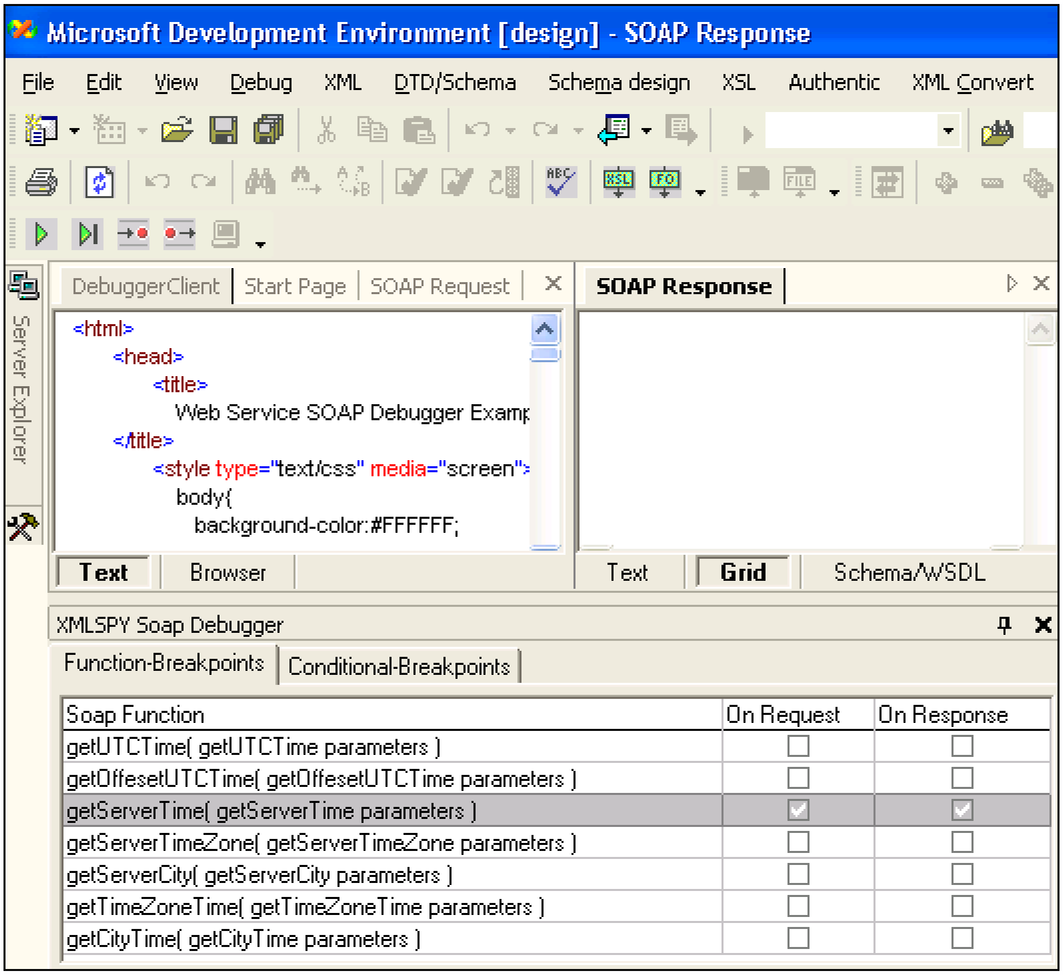
Soap Function	On Request	On Response
getUTCTime(getUTCTime parameters)	□	□

To create a new tab group:

1. Click the **SOAP Response** tab, then drag and drop it somewhere into 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 consisting of the SOAP Response window.



Microsoft Development Environment [design] - SOAP Response
File Edit View Debug XML DTD/Schema Schema design XSL Authentic XML Convert
Server Explorer
DebuggerClient Start Page SOAP Request
SOAP Response
Text Browser
Text Grid Schema/WSDL
XMLSPY Soap Debugger
Function-Breakpoints Conditional-Breakpoints

```
<html>
    <head>
        <title>
            Web Service SOAP Debugger Examp
    </title>
        <style type="text/css" media="screen">
            body{
                background-color:#FFFFFF;
```

Soap Function	On Request	On Response
getUTCTime(getUTCTime parameters)	□	□
getOffesetUTCTime(getOffesetUTCTime parameters)	□	□
getServerTime(getServerTime parameters)	☑	☑
getServerTimeZone(getServerTimeZone parameters)	□	□
getServerCity(getServerCity parameters)	□	□
getTimeZoneTime(getTimeZoneTime parameters)	□	□
getCityTime(getCityTime parameters)	□	□

9 XMLSpy in Eclipse Platform

Eclipse 3.x is an open source framework that integrates different types of applications delivered in form of plugins. XMLSpy for the Eclipse Platform, is an Eclipse Plug-in that allows you to access the functionality of a previously installed XMLSpy Edition from within the Eclipse 3.0 and 3.1 Platform.

Installation Requirements

To successfully install the XMLSpy Plug-in for Eclipse 3.0, or 3.1, you need the following:

- The specific XMLSpy Edition you intend to use: Enterprise, Professional, or Home
- The Eclipse 3.x package, as well as
- The appropriate Java Runtime Edition

Installing the XML Spy Plug-in

To install the XML Spy Plug-in:

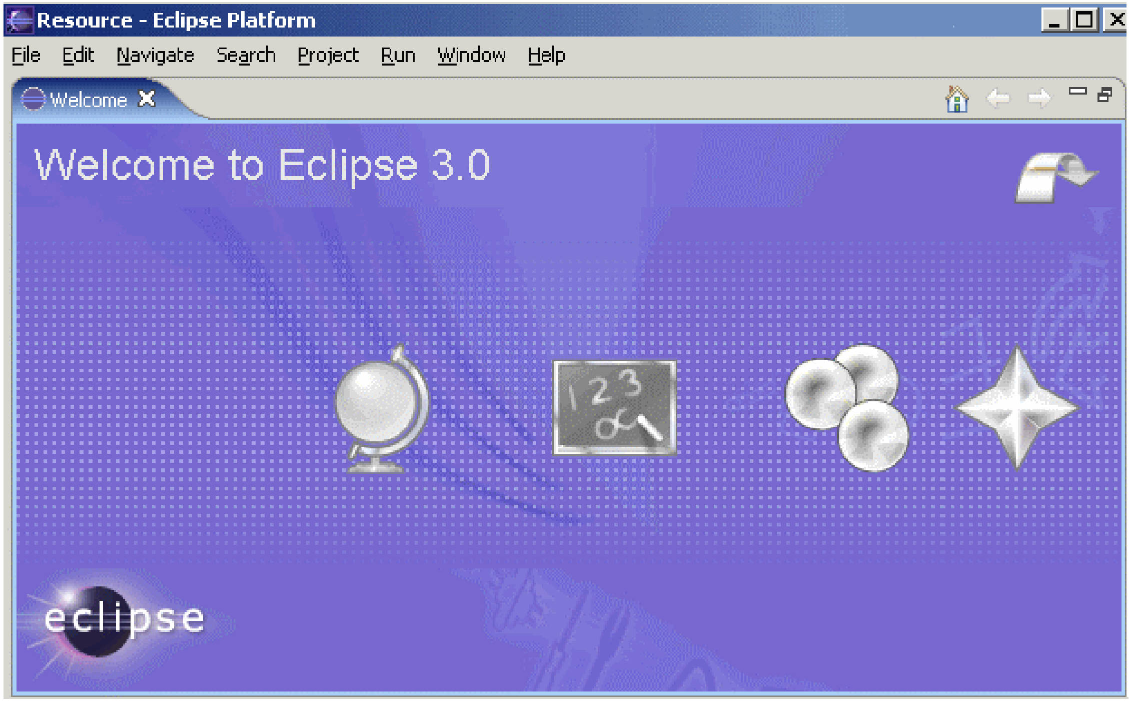
- Download and install the XMLSpy Plugin for Eclipse from the Download section of the Altova.com website. You will be prompted for the installation folder of the Plug-in during the installation process.

The XMLSpy Plug-in for Eclipse supplies the following functionality:

- A fully-featured editor that can edit any type of file that XMLSpy is capable of editing, which also contributes application-specific actions to menu and toolbars.
- A set of Views that define the individual windows of the application: in this case the XMLSpy entry helpers, as well as the individual windows for each of the specific debuggers.
- Different Perspectives that determine the appearance of the workbench. XMLSpy supplies three perspectives: XMLSpy, Debug SOAP and Debug XSLT.
- XMLSpy user help under the menu item **Help | XMLSPY | Table of contents**.

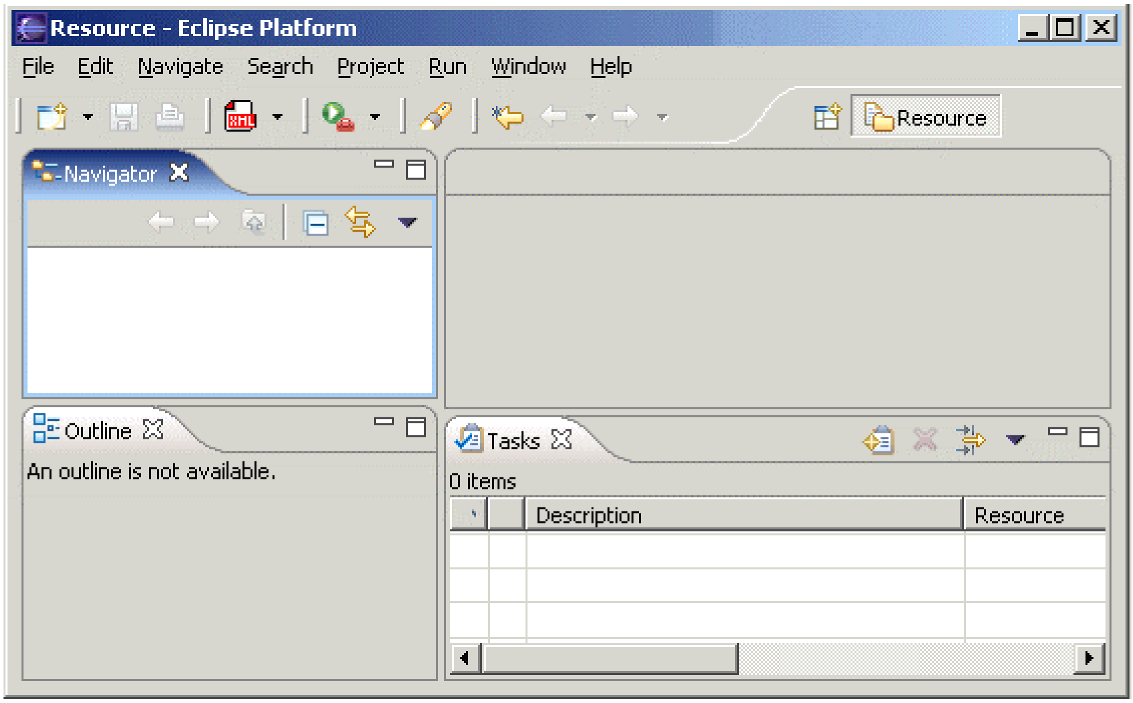
9.1 Starting Eclipse and using the XMLSpy Plug-in

1. Double-click `eclipse.exe` to start the Eclipse Platform.


Resource - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Welcome
Welcome to Eclipse 3.0
eclipse

This opens the "Welcome to Eclipse 3.0" start screen.

2. Place the cursor over the arrow symbol, and click when the "Go To Workbench" text appears. This opens an empty XMLSpy window in Eclipse.


Resource - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Resource
Navigator
Outline
Tasks

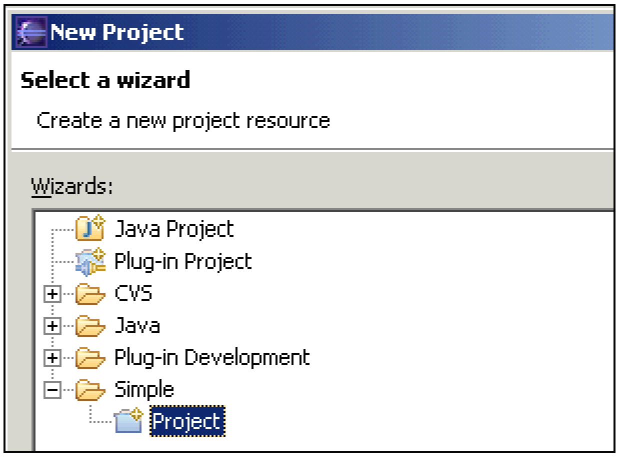
An outline is not available.

0 items

		Description	Resource

Creating a new Project:

1. Right-click in the Navigator window, and select **New | Project | Simple Project**.

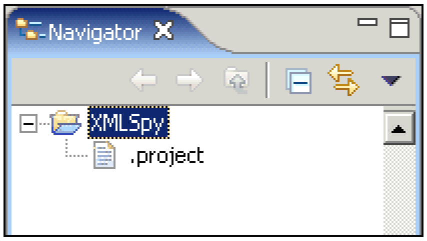


New Project

Select a wizard

Create a new project resource

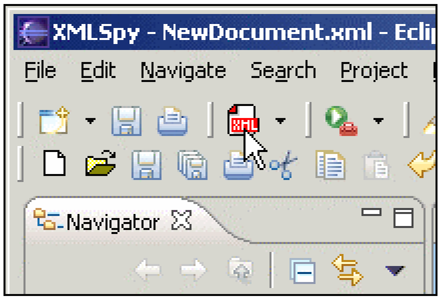
2. Enter `XMLSpy` as the project name, and click **Finish**.
3. This creates the XMLSpy project folder.



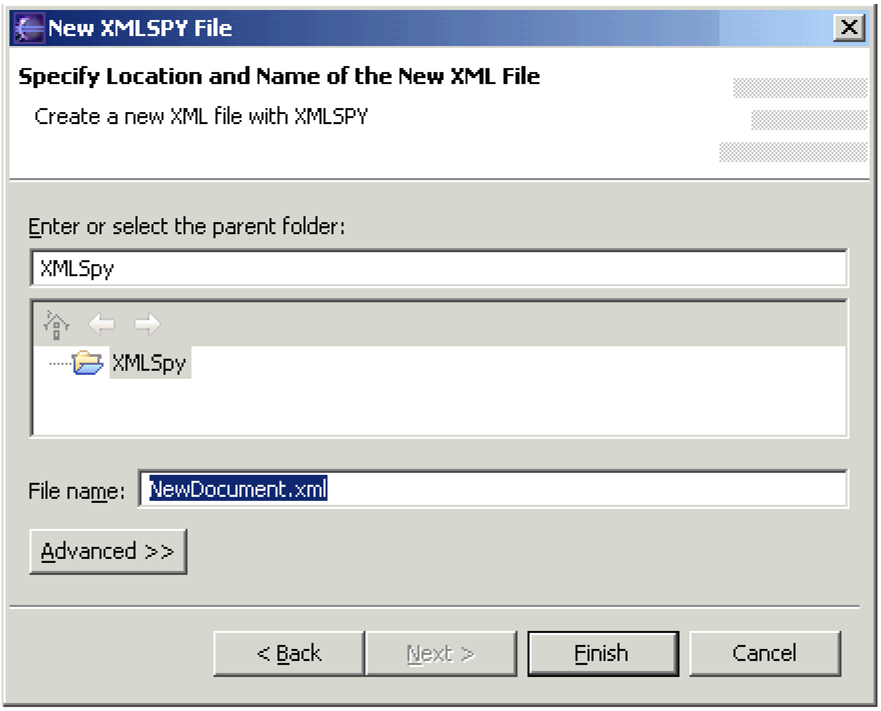
9.2 Creating XML files in Eclipse

To create a new XML file based on schema:

1. Click the XMLSpy XML icon.

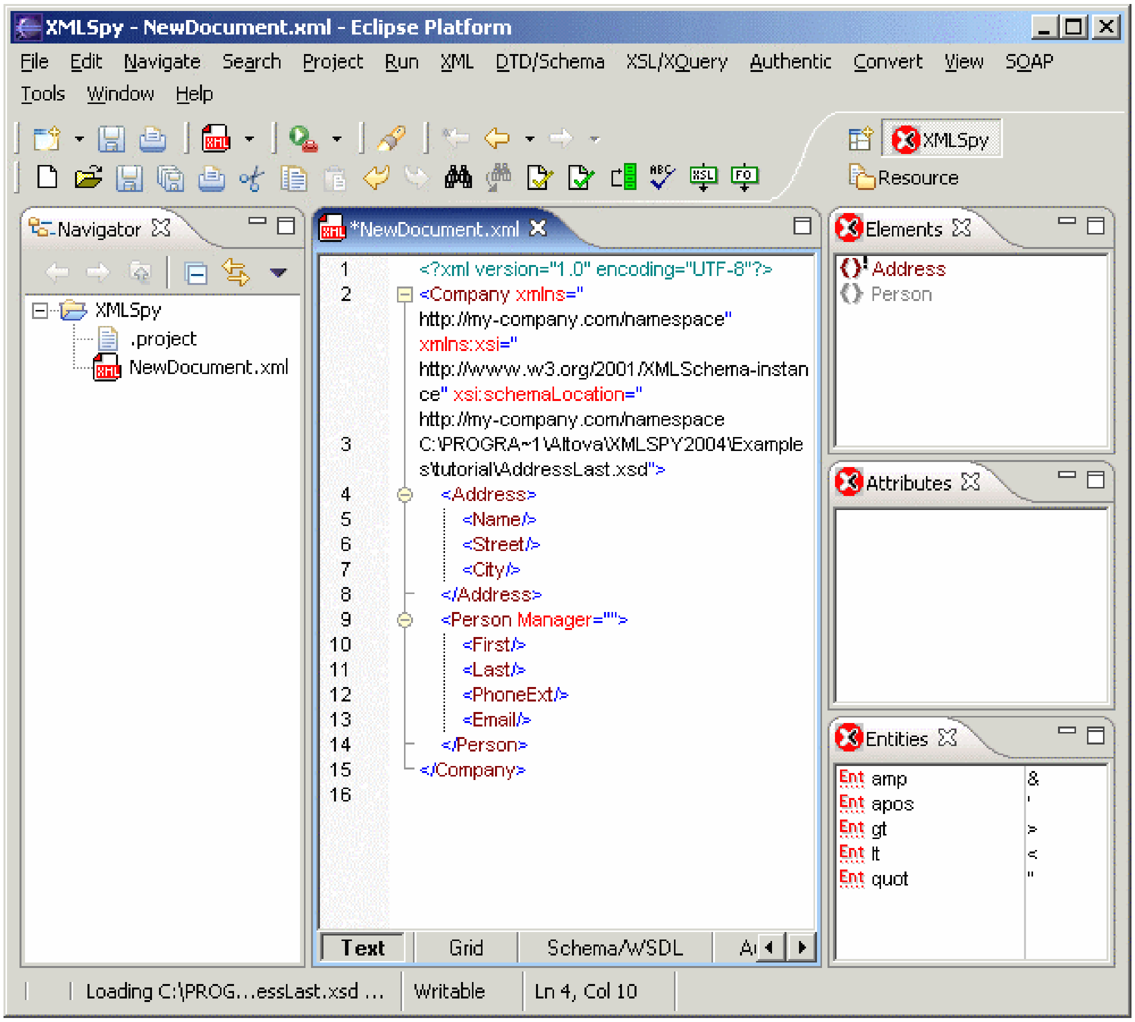

XMLSpy - NewDocument.xml - Ecli
File Edit Navigate Search Project
XML
Navigator

2. Select **xml | XML document** then click **Next**.
   You are then prompted to select a parent folder.


New XMLSPY File
Specify Location and Name of the New XML File
Create a new XML file with XMLSPY
Enter or select the parent folder:
XMLSpy
XMLSpy
File name:
NewDocument.xml
Advanced >>
< Back
Next >
Finish
Cancel

3. Enter `XMLSpy` as the parent folder (or select an existing folder) and use the supplied default filename **NewDocument.xml** and click **Finish**.
   The dialog boxes that now appear are from XMLSpy.
4. Select Schema and click **OK**, then select the schema file using the Browse button (e.g., `AddressLast.xsd`) and click **OK**.
   The new XML file appears in the NewDocument.xml tab in the Text view.

The preconfigured XMLSpy perspective is automatically activated to display the various entry helpers.



```
<?xml version="1.0" encoding="UTF-8"?>
<Company xmlns="
http://my-company.com/namespace"
xmlns:xsi="
http://www.w3.org/2001/XMLSchema-instan
ce" xsi:schemaLocation="
http://my-company.com/namespace
C:\PROGRA~1\Altova\XMLSPY2004\Example
s\tutorial\AddressLast.xsd">
    <Address>
        <Name/>
        <Street/>
        <City/>
    </Address>
    <Person Manager="">
        <First/>
        <Last/>
        <PhoneExt/>
        <Email/>
    </Person>
</Company>
```

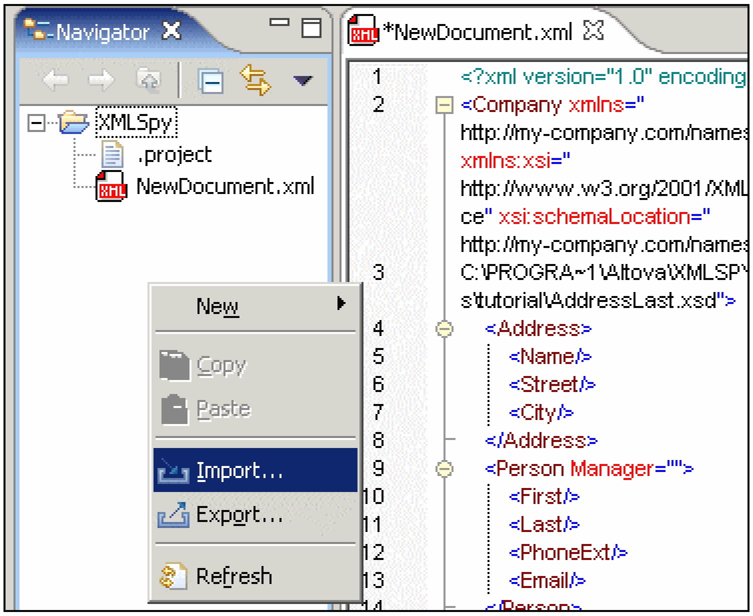
Ent amp	&
Ent apos	'
Ent gt	>
Ent lt	<
Ent quot	"

Loading C:\PROG...essLast.xsd ... | Writable | Ln 4, Col 10

9.3 Importing XML files into folders

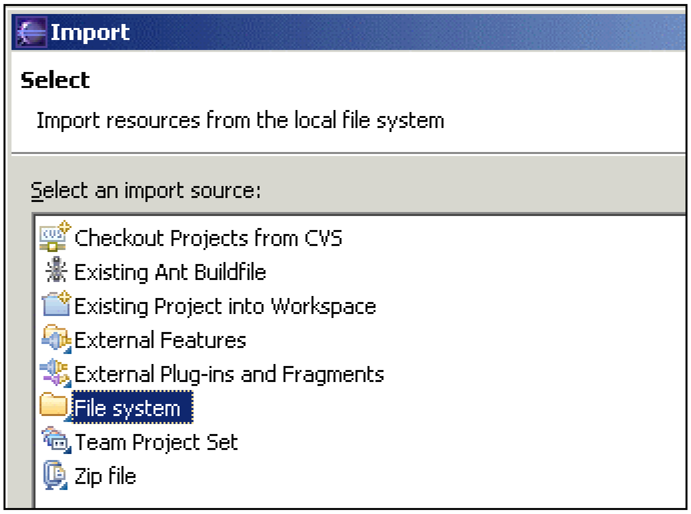
Importing XMLSpy Examples folder into the Navigator:

1. Right-click the **Navigator** tab and click **Import**.



```
<?xml version="1.0" encoding
<Company xmlns="
http://my-company.com/names
xmlns:xsi="
http://www.w3.org/2001/XML
ce" xsi:schemaLocation="
http://my-company.com/names
C:\PROGRA~1\Altova\XMLSPY
s\tutorial\AddressLast.xsd">
  <Address>
    <Name/>
    <Street/>
    <City/>
  </Address>
  <Person Manager="">
    <First/>
    <Last/>
    <PhoneExt/>
    <Email/>
  </Person>
```

2. Select "File system", then click **Next**.



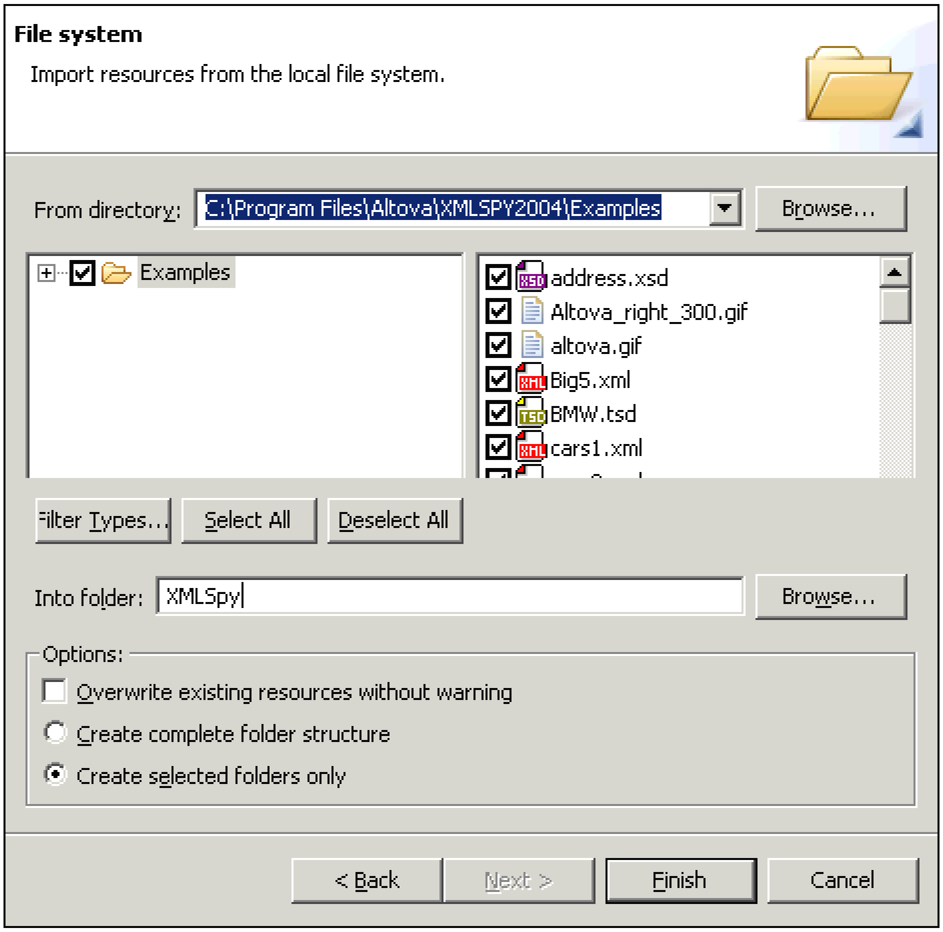
Select

Import resources from the local file system

Select an import source:

Checkout Projects from CVS
Existing Ant Buildfile
Existing Project into Workspace
External Features
External Plug-ins and Fragments
File system
Team Project Set
Zip file

3. Click the **Browse** button to the right of the "From directory:" text box, and select the Examples directory in your XMLSpy folder.



File system

Import resources from the local file system.

From directory: C:\Program Files\Altova\XMLSPY2004\Examples

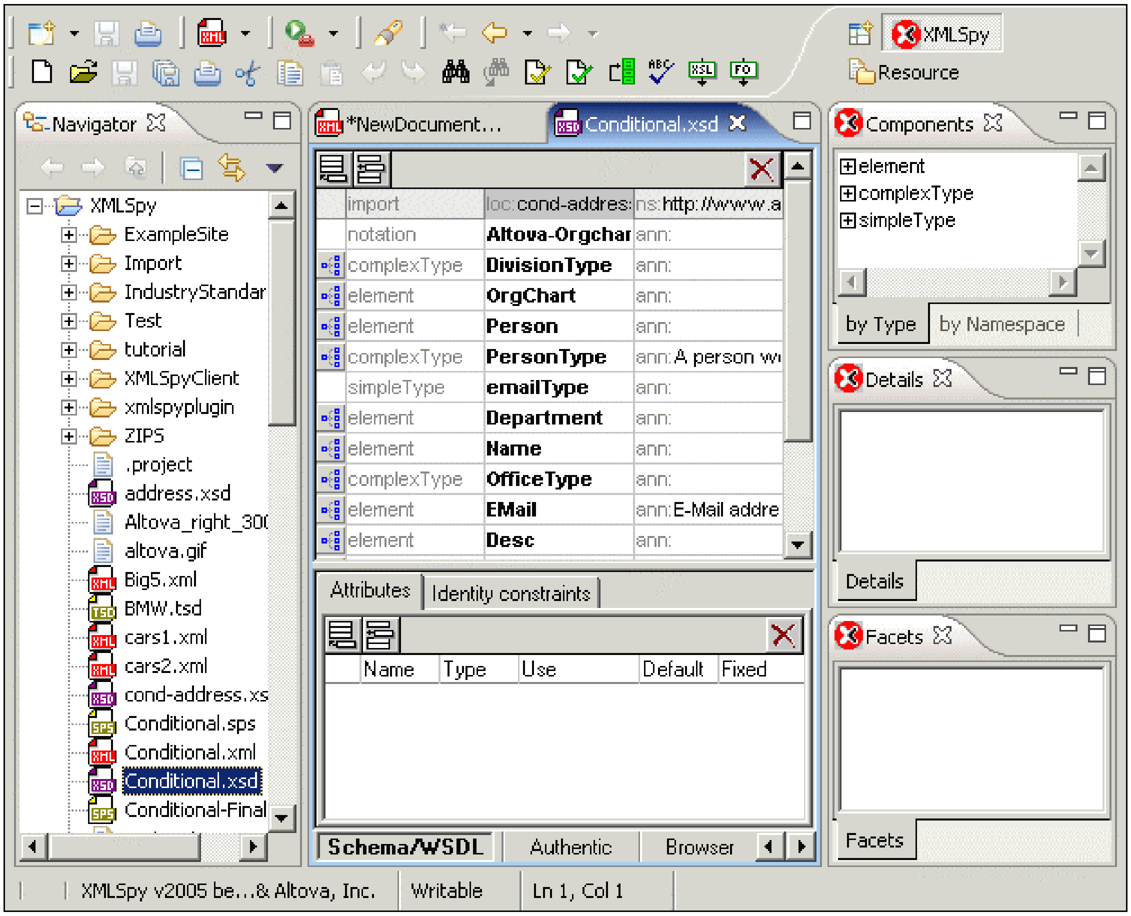
☑ Examples

☑ address.xsd
☑ Altova_right_300.gif
☑ altova.gif
☑ Big5.xml
☑ BMW.tsd
☑ cars1.xml

Into folder: XMLSpy

Options:
□ Overwrite existing resources without warning
○ Create complete folder structure
● Create selected folders only

4. Activate the **Examples** check box.
   This activates all files in the various subdirectories in the window at right.
5. Click the **Browse** button, next to the "Into folder:" text box, to select the target folder, then click **Finish**.
   The selected folder structure and files will be copied into the Eclipse workspace.
6. Double-click a file in Navigator to open it (e.g., `Conditional.xsd`).



import	loc:cond-addres:	ns:http://www.a
notation	**Altova-Orgchar**	ann:
complexType	**DivisionType**	ann:
element	**OrgChart**	ann:
element	**Person**	ann:
complexType	**PersonType**	ann: A person wi
simpleType	**emailType**	ann:
element	**Department**	ann:
element	**Name**	ann:
complexType	**OfficeType**	ann:
element	**EMail**	ann: E-Mail addre
element	**Desc**	ann:

Name	Type	Use	Default	Fixed

9.4 Differences between Eclipse and standalone versions

The Enterprise, Professional and Home editions of the Eclipse Plug-in for XMLSpy generally have the same functionality as their standalone counterparts.

Unsupported features in the integrated version:

Info window
The Info window is not supported. This window gives extra information on currently selected elements/attributes etc. E.g. the name of the element or attribute, the datatype, enumerations and occurrence.

Project window and **Project** menu (as well as source control submenus).
The Project menu (and Source control submenus) are not available, use the Eclipse Navigator.

Tools | XML Spy Options | Scripting
The scripting environment is currently not available.

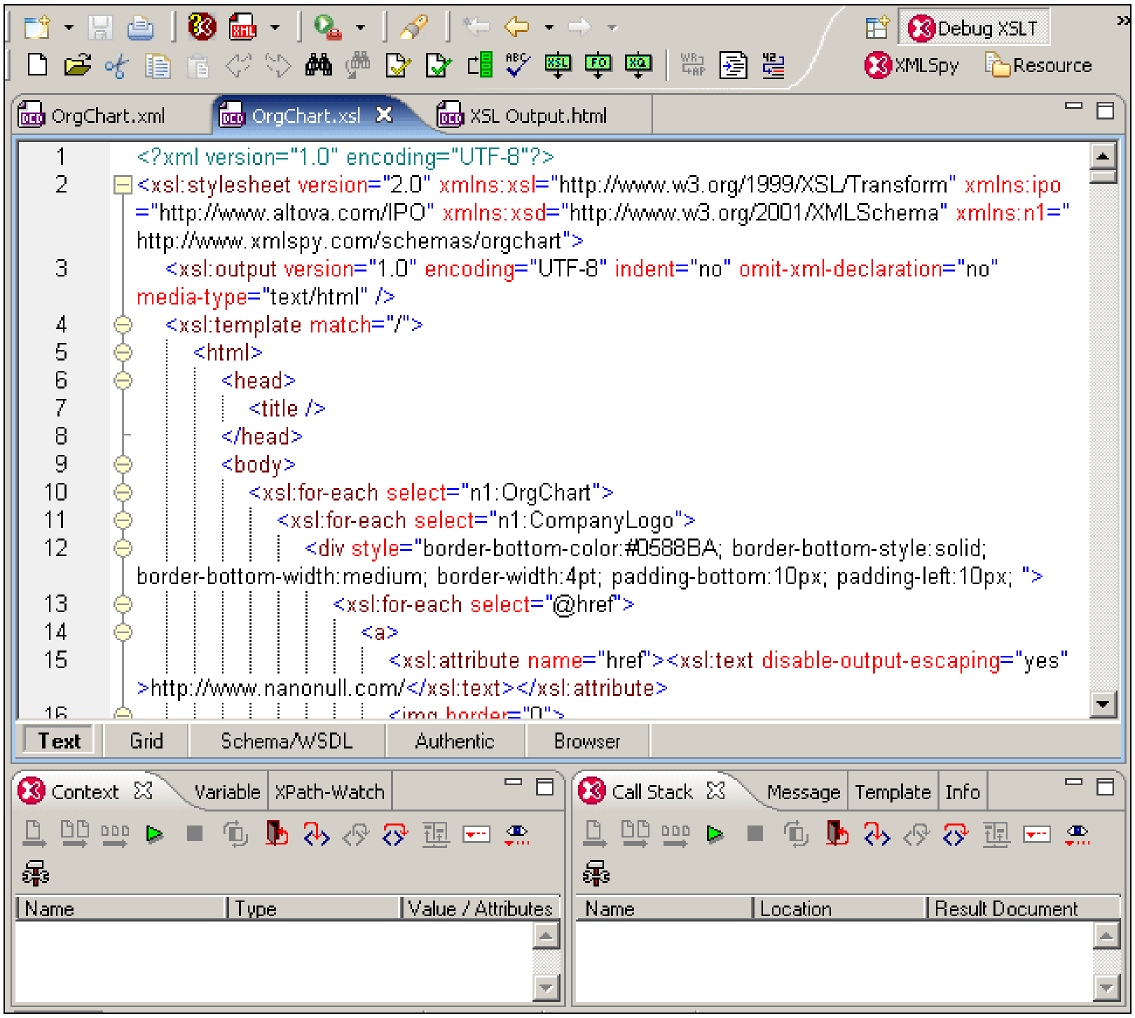
Separate Browser window
The "Show in separate window by default" check box in the Options dialog (**Tools | Options**, **View** tab) is not supported. This means that the Text and Browser view are always incorporated in the same window when you transform an XML file to HTML file for example.

Authentic view
Text state icons are not available in the Authentic view. The functions or formatting that they may provide, are still available in the entry helper or context menu, however. Please see the Stylesheet Designer documentation for more information.

9.5 Eclipse Platform and XSLT Debugger

To debug XSLT:

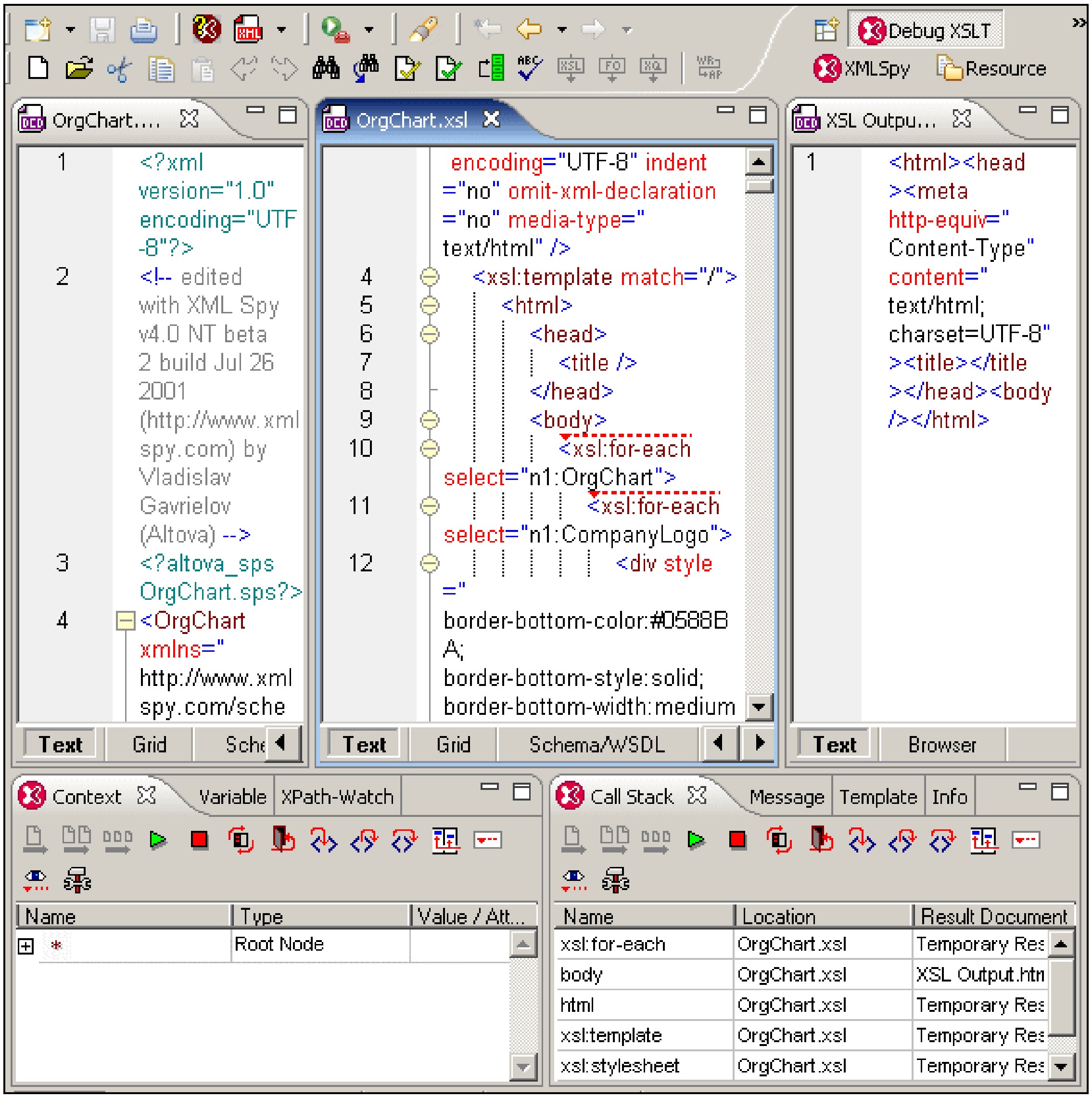
1. Open the **OrgChart.xml** and **Orgchart.xsl** files.
2. Select the menu option **XSL/XQuery | Start Debugger/Go**.
3. Choose the sample XML file you want to use (click the Window button and select OrgChart.xml).
   A prompt appears stating that the view will be changed; click **OK** to proceed.



```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:ipo
="http://www.altova.com/IPO" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:n1="
http://www.xmlspy.com/schemas/orgchart">
    <xsl:output version="1.0" encoding="UTF-8" indent="no" omit-xml-declaration="no"
media-type="text/html" />
    <xsl:template match="/">
        <html>
            <head>
                <title />
            </head>
            <body>
                <xsl:for-each select="n1:OrgChart">
                    <xsl:for-each select="n1:CompanyLogo">
                        <div style="border-bottom-color:#0588BA; border-bottom-style:solid;
border-bottom-width:medium; border-width:4pt; padding-bottom:10px; padding-left:10px; ">
                            <xsl:for-each select="@href">
                                <a>
                                    <xsl:attribute name="href"><xsl:text disable-output-escaping="yes"
>http://www.nanonull.com/</xsl:text></xsl:attribute>
                                    <img border="0">
```

Creating separate file windows:

1. Click the **XSL Output.html** tab, drag to the far right, and drop when an arrow appears. This decouples the tab from the tab group.
   Window outlines are displayed while dragging, to help you when positioning them.
2. Use the same method to decouple the **OrgChart.xsl** tab and place it between the two windows.
   The graphic below shows the new layout, and the debugging process in action.



```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.0 NT beta 2 build Jul 26 2001 (http://www.xmlspy.com) by Vladislav Gavrielov (Altova) -->
<?altova_sps OrgChart.sps?>
<OrgChart xmlns="http://www.xmlspy.com/sche
```

```
 encoding="UTF-8" indent="no" omit-xml-declaration="no" media-type="text/html" />
  <xsl:template match="/">
    <html>
      <head>
        <title />
      </head>
      <body>
        <xsl:for-each select="n1:OrgChart">
          <xsl:for-each select="n1:CompanyLogo">
            <div style="border-bottom-color:#0588BA; border-bottom-style:solid; border-bottom-width:medium
```

```
<html><head><meta http-equiv="Content-Type" content="text/html; charset=UTF-8"><title></title></head><body/></html>
```

Name	Type	Value / Att...
*	Root Node	

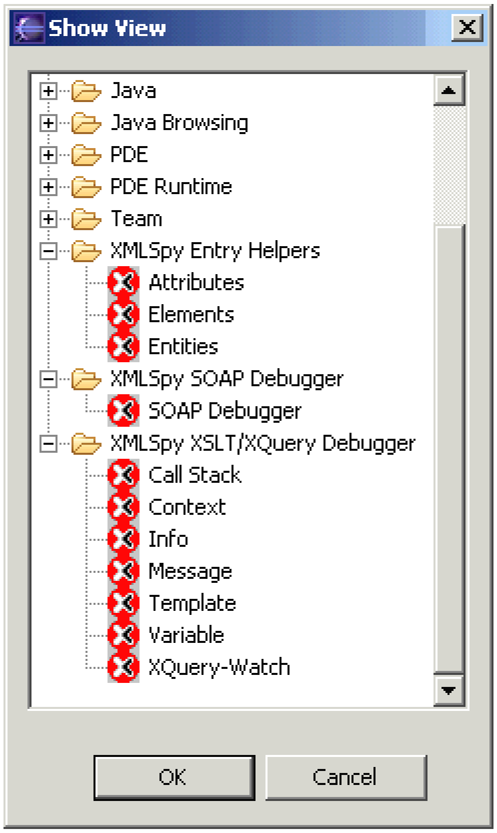
Name	Location	Result Document
xsl:for-each	OrgChart.xsl	Temporary Res
body	OrgChart.xsl	XSL Output.htm
html	OrgChart.xsl	Temporary Res
xsl:template	OrgChart.xsl	Temporary Res
xsl:stylesheet	OrgChart.xsl	Temporary Res

Closing the debug session, automatically closes the XSL `Output.html` file and switches back to the XMLSpy perspective.

9.6 Eclipse views and perspectives

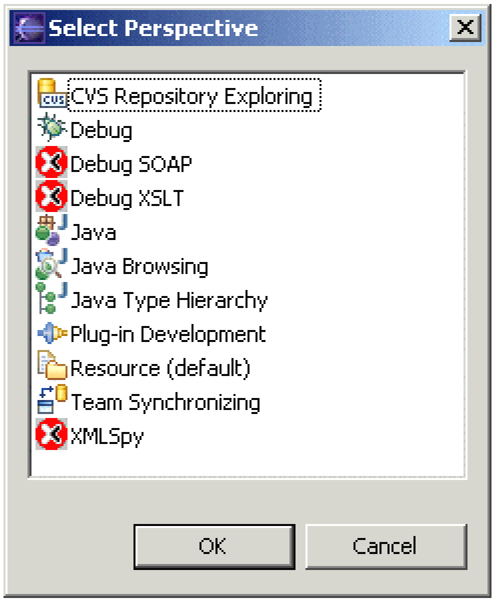
The XMLSpy Views define the main entry helpers as well as the debugger windows for each specific debugger.

- Select the menu option **Window | Show View | Other...** to display the currently available views.



The XMLSpy Perspectives define the main application window as well as the debugger user interfaces.

- Select the menu option **Window | Open Perspective| Other...** to display the currently available perspectives.



Select Perspective

- CVS Repository Exploring
- Debug
- Debug SOAP
- Debug XSLT
- Java
- Java Browsing
- Java Type Hierarchy
- Plug-in Development
- Resource (default)
- Team Synchronizing
- XMLSpy

10 XSLT and XQuery Debugger

The XSLT/XQuery Debugger enables you to test and debug XSLT stylesheets and XQuery documents. The XSLT/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/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 `version` attribute of `xsl:stylesheet` (or `xsl:transform`) element of XSLT stylesheet.

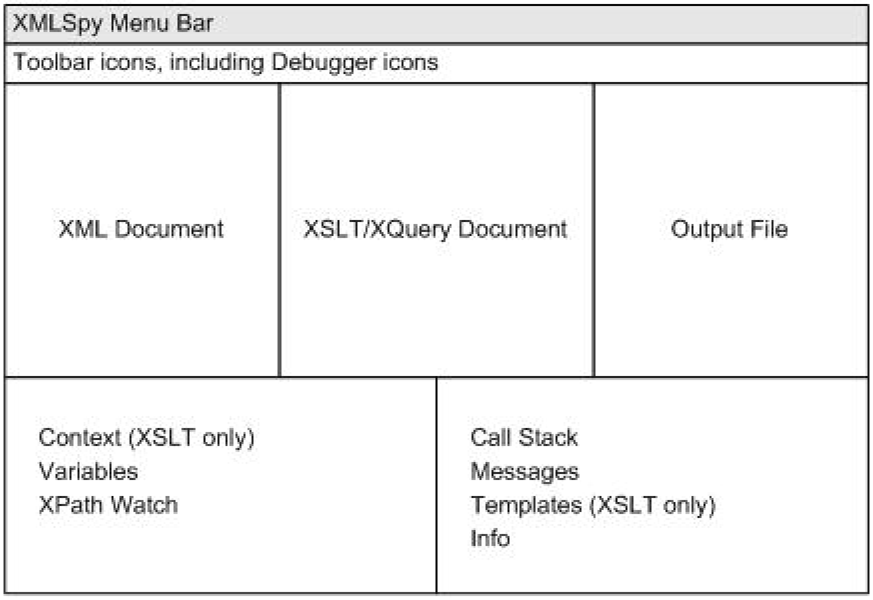
For details about the three Altova engines, please see the following sections:

- Altova XSLT 1.0 Engine
- Altova XSLT 2.0 Engine
- Altova XQuery 1.0 Engine

10.1 Mechanism and Interface

The broad mechanism used for debugging XSLT and XQuery files using the XSLT/XQuery Debugger is as follows:

- Open a debugging session. 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/XQuery Debugger works only in Text View and Enhanced Grid View. If the view of the active document is not Text View or Enhanced 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/XQuery Debugger. You can also choose to have this option set permanently.
- Step through the XSLT or XQuery document. 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.



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/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/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.

10.2 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)

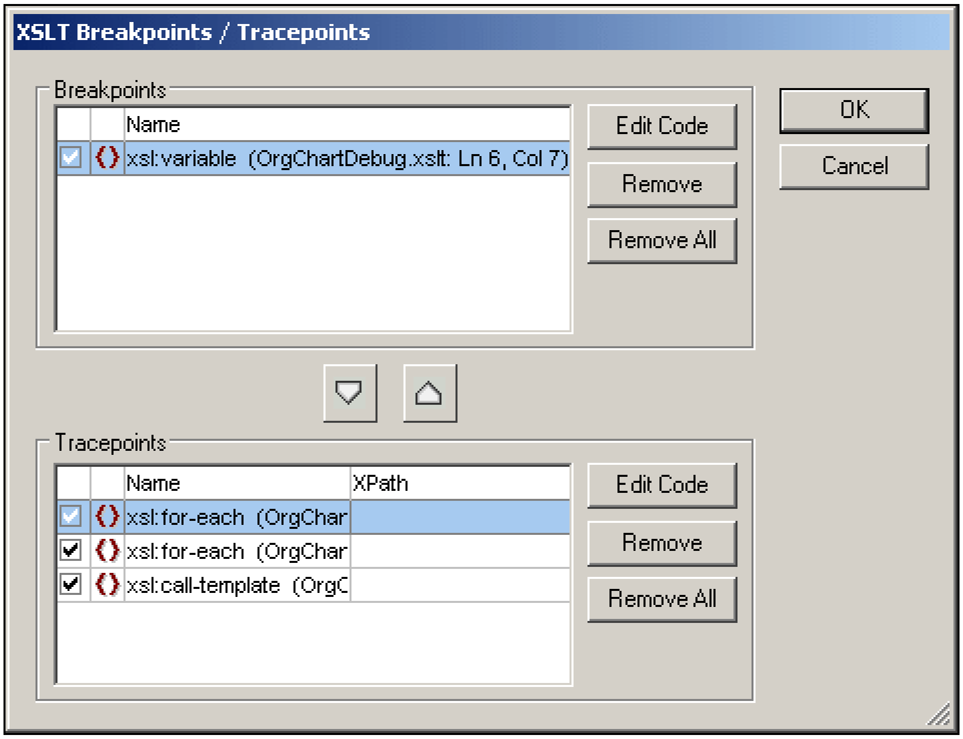
This command (no toolbar icon exists) enables or disables already defined tracepoints. The blue tracepoint highlight turns to gray when a tracepoint is disabled. No output is made in the Trace window for disabled tracepoints. To disable/enable a tracepoint, place the cursor in that node name and click 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.



XSLT Breakpoints / Tracepoints

Breakpoints

		Name
☑	{}	xsl:variable (OrgChartDebug.xslt: Ln 6, Col 7)

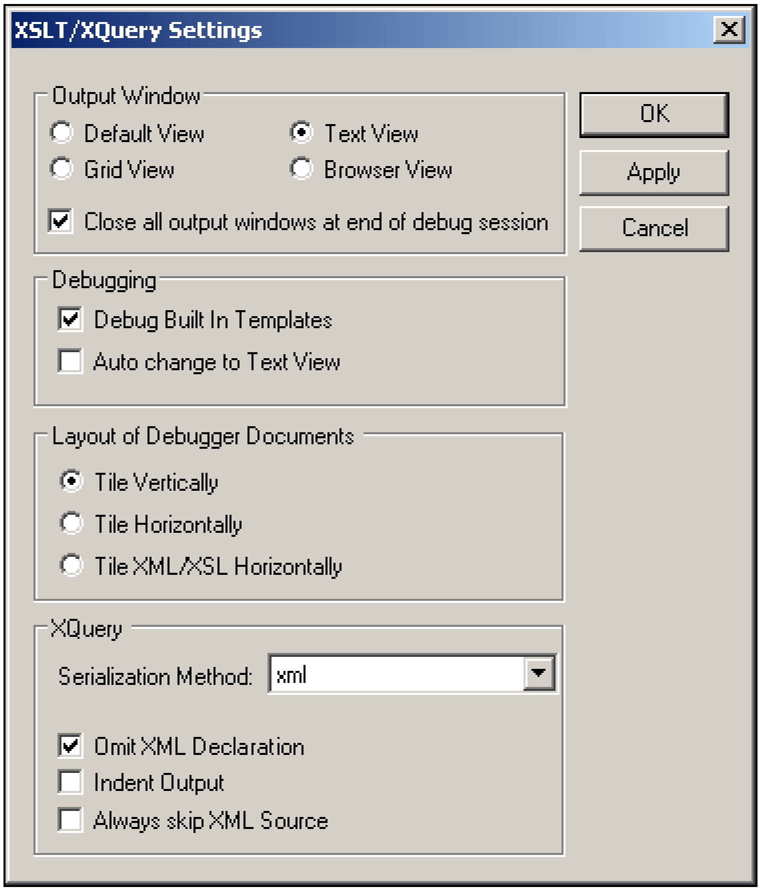
Tracepoints

		Name	XPath
☑	{}	xsl:for-each (OrgChar	
☑	{}	xsl:for-each (OrgChar	
☑	{}	xsl:call-template (OrgC	

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.

10.3 Settings

The XSLT/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

○ Default View ● Text View
○ Grid View ○ Browser View

☑ Close all output windows at end of debug session

Debugging

☑ Debug Built In Templates
☐ Auto change to Text View

Layout of Debugger Documents

● Tile Vertically
○ Tile Horizontally
○ Tile XML/XSL Horizontally

XQuery

Serialization Method: xml

☑ Omit XML Declaration
☐ Indent Output
☐ Always skip XML Source

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.

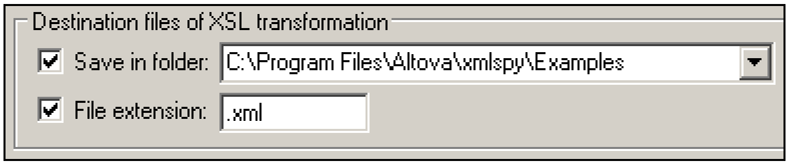
The XSLT Debugger works only in Text View or Grid View. The Auto Change to Text View option enables you to automatically switch to the Text View of a document for debugging if a document is not in Text View or Grid View. (The XQuery Debugger works in Text View only.)

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.

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.



Destination files of XSL transformation

☑ Save in folder: C:\Program Files\Altova\xmlspy\Examples

☑ File extension: .xml

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.

10.4 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:

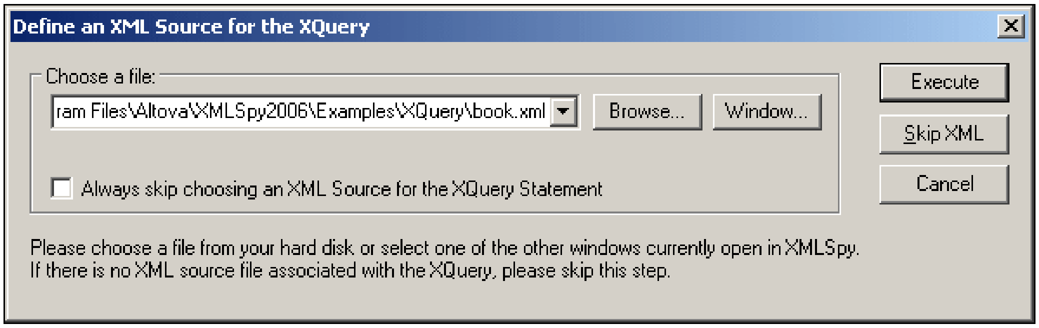
- In XML files: Open the file, click **XSL/XQuery | Assign XSL**, and select the XSLT file.
- In XSLT files: Open the file, click **XSL/XQuery | Assign sample XML file...**, and select the XML file.

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.



ram Files\Altova\XMLSpy2006\Examples\XQuery\book.xml

□ Always skip choosing an XML Source for the XQuery Statement

Please choose a file from your hard disk or select one of the other windows currently open in XMLSpy.
If there is no XML source file associated with the XQuery, please skip this step.

(The dialog shown in the screenshot appears when you start a debugging session from an XQuery file.)

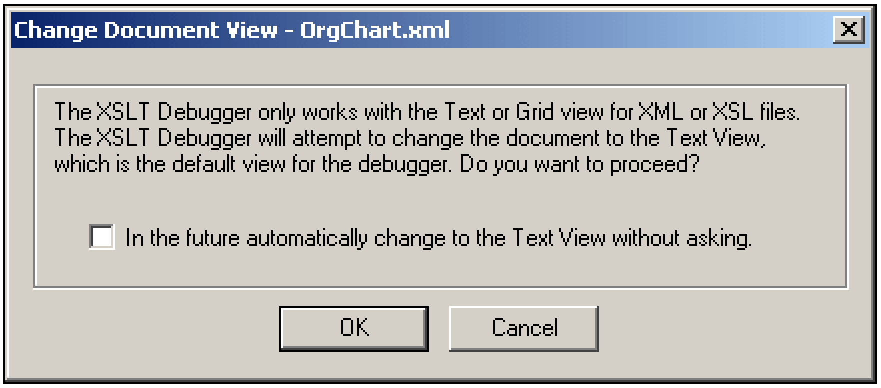
After you select the required associated file or skip an optional association, the debugging session is started.

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/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.



Change Document View - OrgChart.xml

The XSLT Debugger only works with the Text or Grid view for XML or XSL files. The XSLT Debugger will attempt to change the document to the Text View, which is the default view for the debugger. Do you want to proceed?

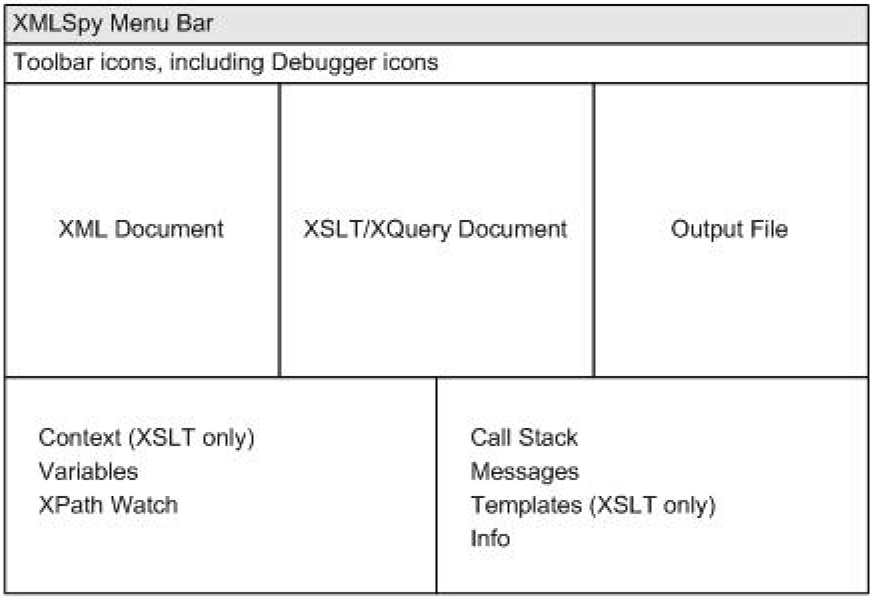
□ In the future automatically change to the Text View without asking.

Clicking **OK** causes the document to open in Text View. Note that XQuery files are always displayed in Text View.

10.5 Information Windows

Information windows that are opened in the XSLT/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/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/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

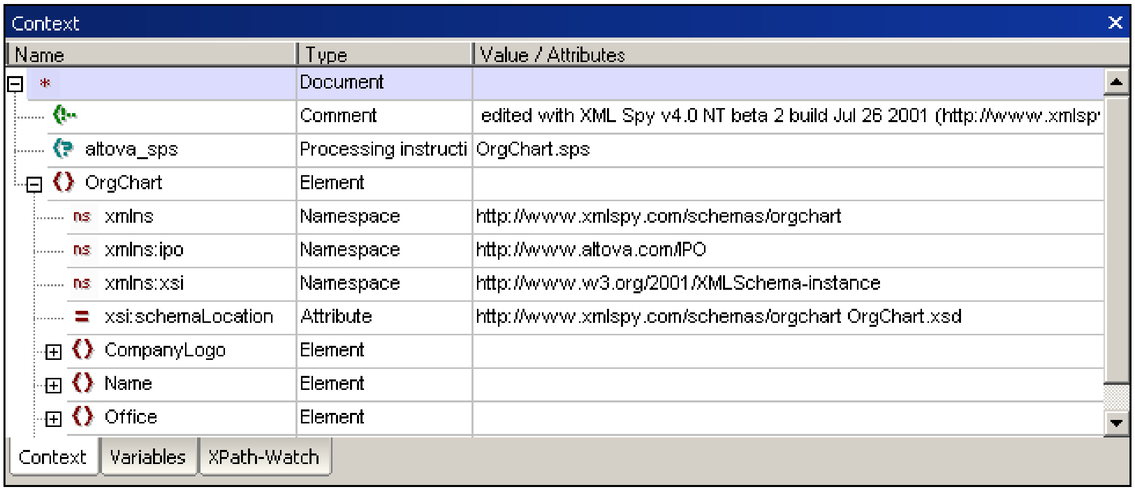
In the default layout, therefore, there are two window groups, each having tabs for the different windows in them. One tab is active at a time. So, for example, to display information about Variables in the first information window group, click the **Variables** tab. This causes the Variables information 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.

10.5.1 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.

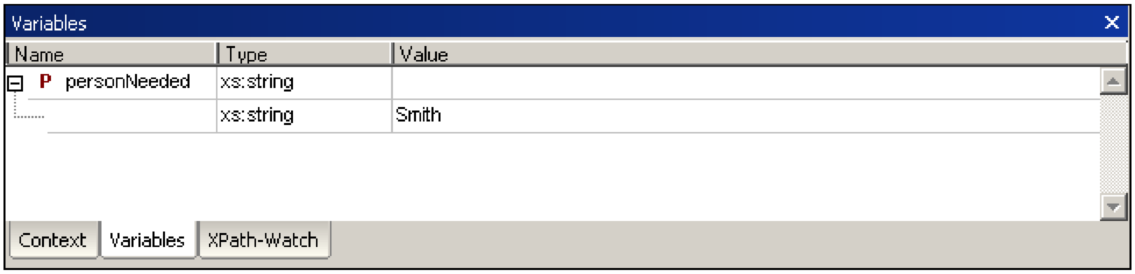


Name	Type	Value / Attributes
⊟ *	Document	
	Comment	edited with XML Spy v4.0 NT beta 2 build Jul 26 2001 (http://www.xmlsp
altova_sps	Processing instructi	OrgChart.sps
⊟ OrgChart	Element	
ns xmlns	Namespace	http://www.xmlspy.com/schemas/orgchart
ns xmlns:ipo	Namespace	http://www.altova.com/IPO
ns xmlns:xsi	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.

10.5.2 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.



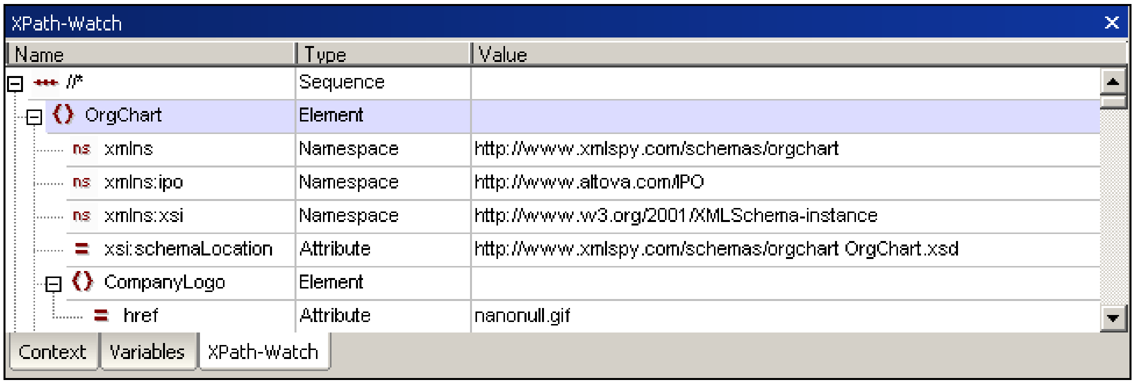
Name	Type	Value
⊟ P personNeeded	xs:string	
	xs:string	Smith

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.

10.5.3 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.



Name	Type	Value
//*	Sequence	
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	
href	Attribute	nanonull.gif

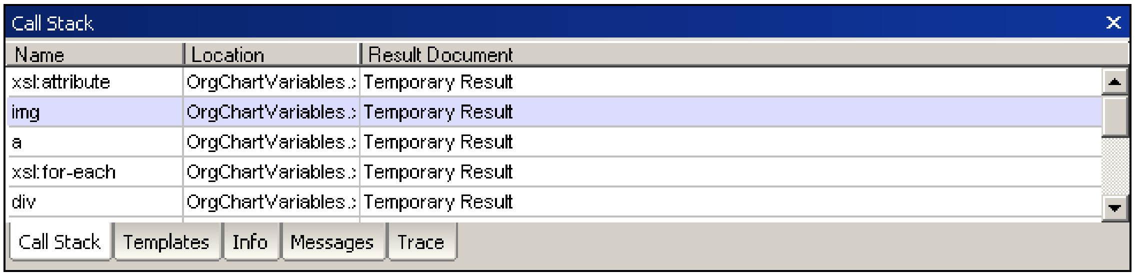
To enter an XPath expression, double-click in the text field under the Name column and enter the XPath. Use expressions that are correct according to the XPath version that corresponds to the XSLT version of the XSLT stylesheet (XPath 1.0 for XSLT 1.0, and XPath 2.0 for XSLT 2.0).

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.

10.5.4 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.



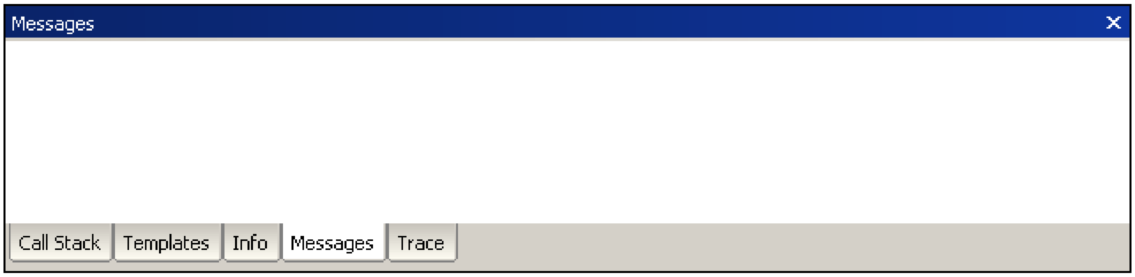
Name	Location	Result Document
xsl:attribute	OrgChartVariables.:	Temporary Result
img	OrgChartVariables.:	Temporary Result
a	OrgChartVariables.:	Temporary Result
xsl:for-each	OrgChartVariables.:	Temporary Result
div	OrgChartVariables.:	Temporary Result

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.

10.5.5 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.

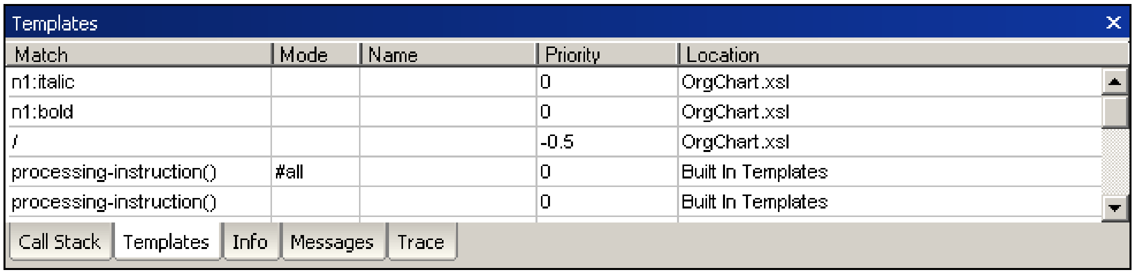


XQuery
In XQuery debugging sessions, the Messages Window displays error messages.

10.5.6 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.



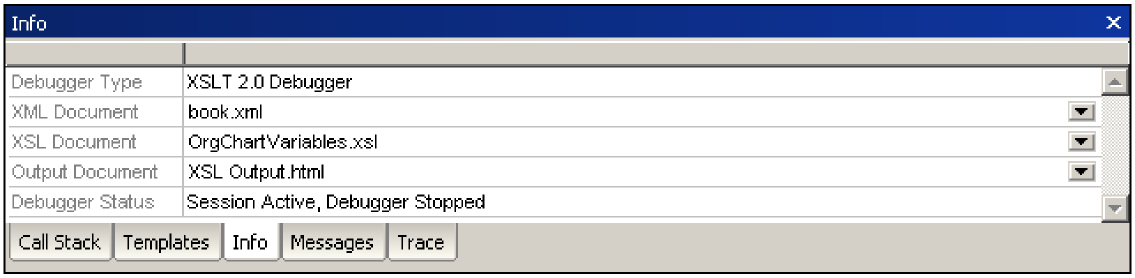
Match	Mode	Name	Priority	Location
n1:italic			0	OrgChart.xsl
n1:bold			0	OrgChart.xsl
/			-0.5	OrgChart.xsl
processing-instruction()	#all		0	Built In Templates
processing-instruction()			0	Built In Templates

In the screenshot above, there are two templates in the XSLT stylesheet: a template which matches the document node `/`, and a template named `tokenize`. 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.

10.5.7 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.

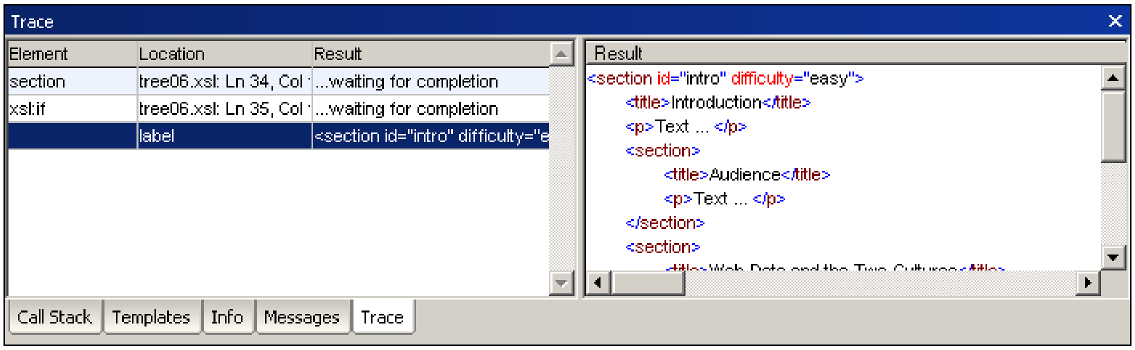


Info

Debugger Type	XSLT 2.0 Debugger
XML Document	book.xml
XSL Document	OrgChartVariables.xsl
Output Document	XSL Output.html
Debugger Status	Session Active, Debugger Stopped

10.5.8 Trace Window

The Trace Window is displayed in XSLT and XQuery debugging sessions.



Trace

Element	Location	Result
section	tree06.xsl: Ln 34, Col '	...waiting for completion
xsl:if	tree06.xsl: Ln 35, Col '	...waiting for completion
	label	<section id="intro" difficulty="e

Result

```
<section id="intro" difficulty="easy">
    <title>Introduction</title>
    <p>Text ... </p>
    <section>
        <title>Audience</title>
        <p>Text ... </p>
    </section>
    <section>
```

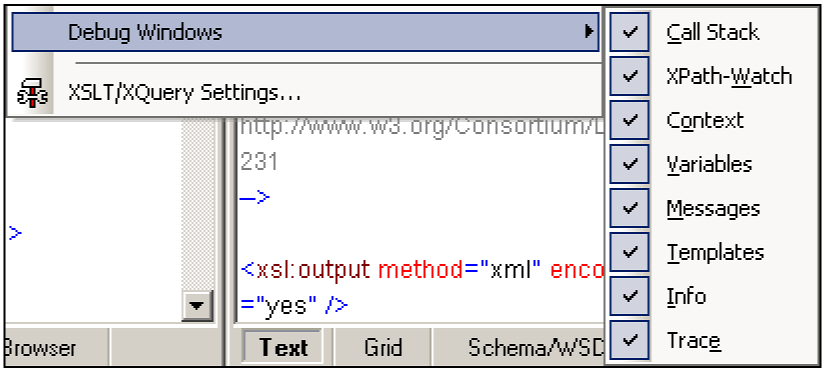
The Trace Window contains the element the tracepoint is set for, its location in the XSLT stylesheet and the result generated when that element is executed. Click on a row in the left side of the window to display the full result on the right.

10.5.9 Arranging the Information Windows

The Information Windows can be arranged inside the XSLT/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

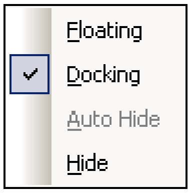
In the **XSL/XQuery** menu, placing the cursor over the item **Debug Windows** pops up the list of Info Windows. You can hide or show individual windows by clicking the window.



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.

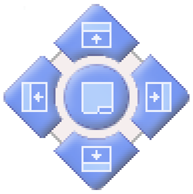


Click the required option to cause that window to float, be docked or be hidden.

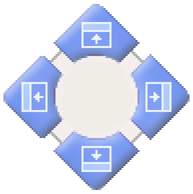
Drag-and-drop
You can drag a window by its tab or title bar and place it at a desired location.

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.

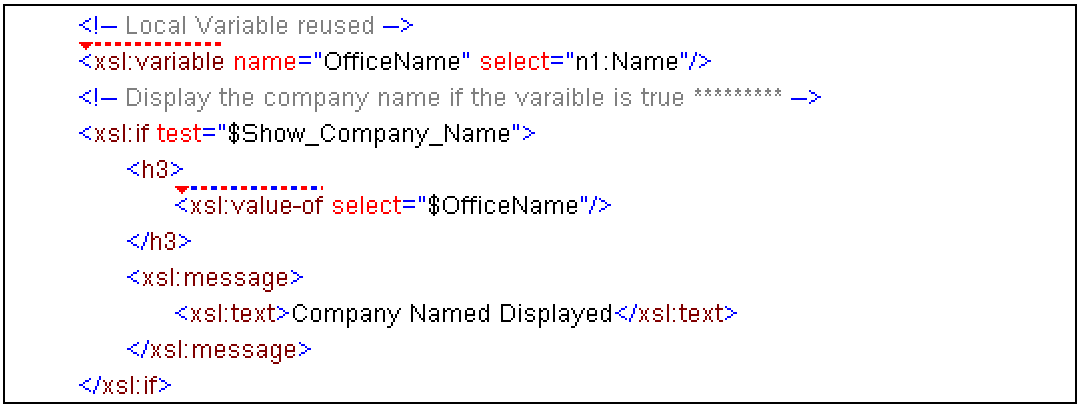


You can also double-click the title bar of a window to toggle it between its docked and floating positions.

10.6 Breakpoints

The XSLT/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 varaible 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.

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).

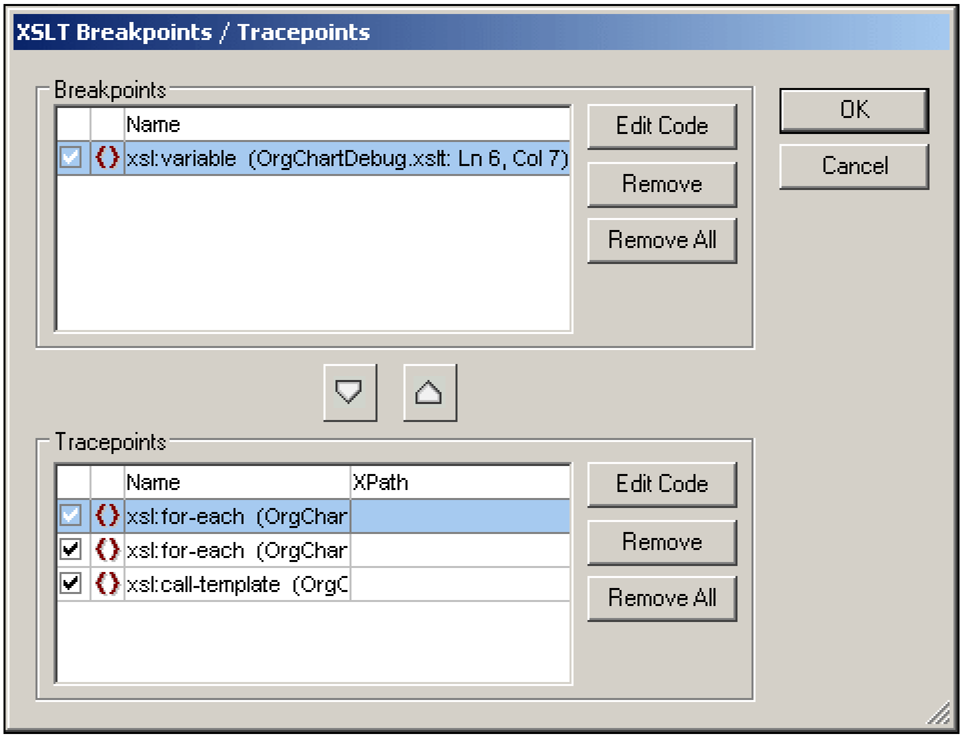
Disabling/enabling breakpoints:
After inserting breakpoints, you can disable them if you wish to skip over breakpoints without having to delete them. You can enable them again when necessary.

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.



XSLT Breakpoints / Tracepoints

Breakpoints

		Name
☑	{}	xsl:variable (OrgChartDebug.xslt: Ln 6, Col 7)

Tracepoints

		Name	XPath
☑	{}	xsl:for-each (OrgChar	
☑	{}	xsl:for-each (OrgChar	
☑	{}	xsl:call-template (OrgC	

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.

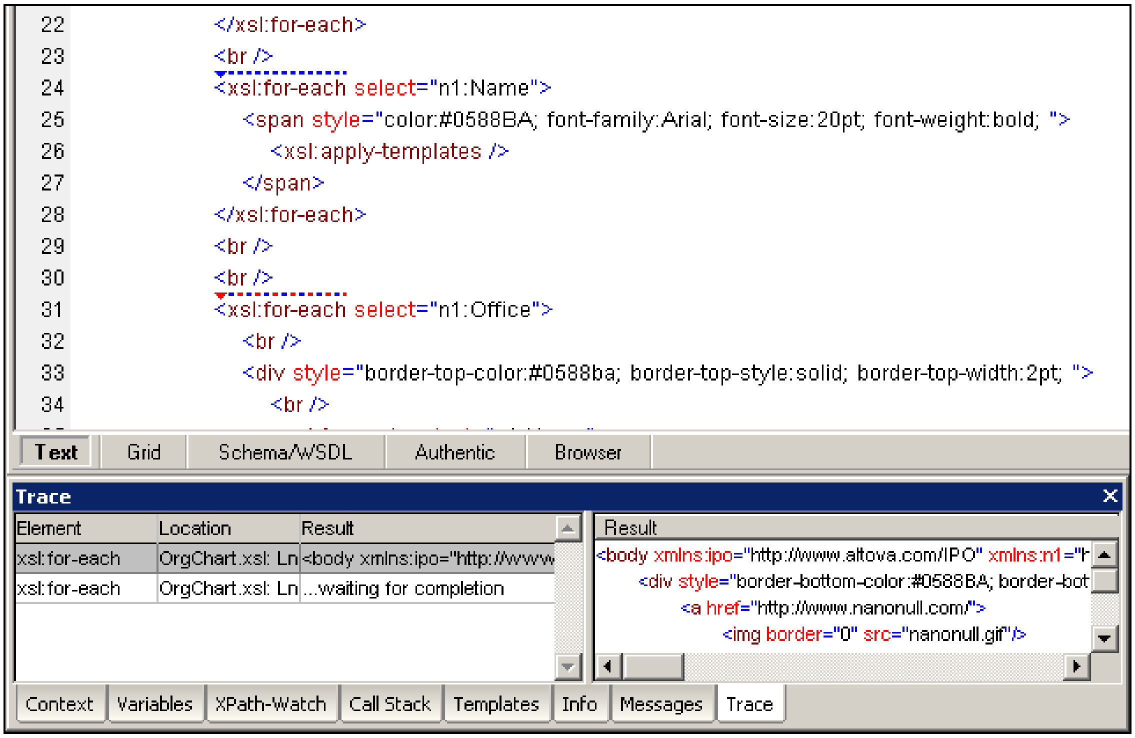
10.7 Tracepoints

The XSLT/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*).



```
</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">
   <br />
   <div style="border-top-color:#0588ba; border-top-style:solid; border-top-width:2pt; ">
      <br />
```

Element	Location	Result
xsl:for-each	OrgChart.xsl: Ln	<body xmlns:ipo="http://www
xsl:for-each	OrgChart.xsl: Ln	...waiting for completion

Result

```
<body xmlns:ipo="http://www.altova.com/IPO" xmlns:n1="h
   <div style="border-bottom-color:#0588BA; border-bot
      <a href="http://www.nanonull.com/">
         <img border="0" src="nanonull.gif"/>
```

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**.

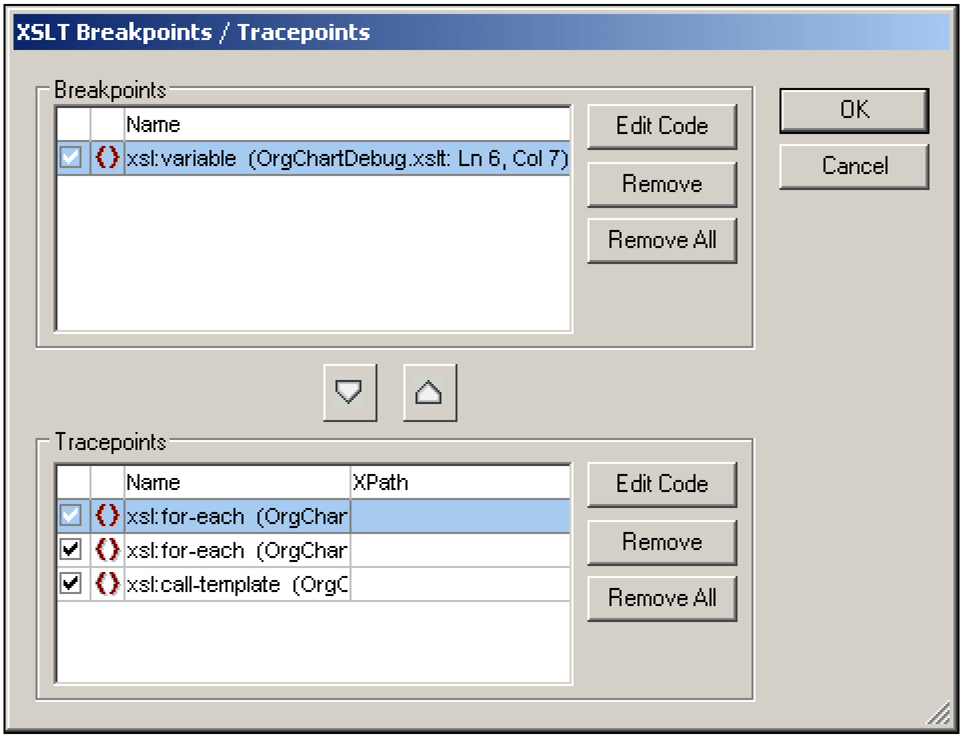
To remove a tracepoint:

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**.



Breakpoints

		Name
☑	{}	xsl:variable (OrgChartDebug.xslt: Ln 6, Col 7)

Tracepoints

		Name	XPath
☑	{}	xsl:for-each (OrgChar	
☑	{}	xsl:for-each (OrgChar	
☑	{}	xsl:call-template (OrgC	

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

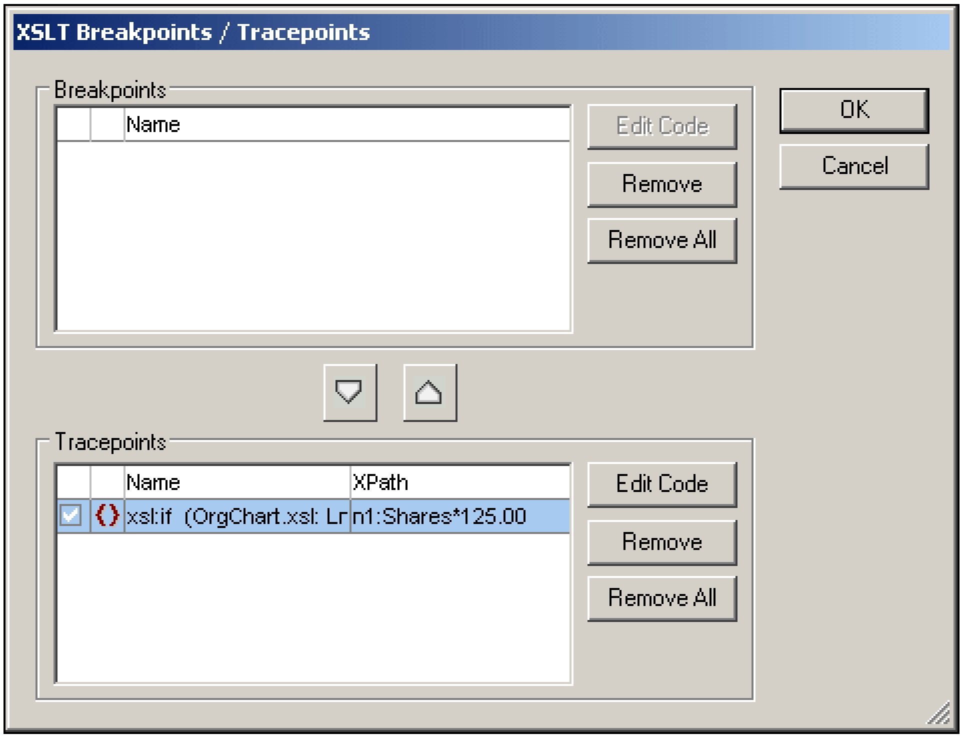
tracepoints in all open XSLT stylesheet documents.
2. Enter the XPath in the **XPath** column in the row that corresponds to the tracepoint.

Example:

The following example uses the file `OrgChart.xsl`, 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).

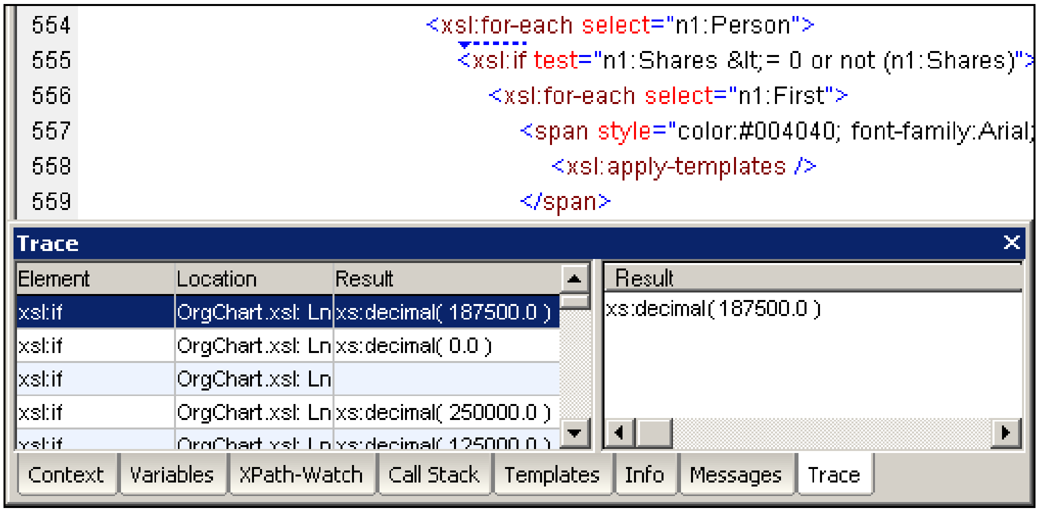
Do the following:

1. Open the file `OrgChart.xsl`.
2. Set a tracepoint at line 555.
3. Open the XSLT Breakpoints / Tracepoints dialog and enter the XPath `n1:Shares*125.00` for the tracepoint you just set.



		Name	XPath
☑	{}	xsl:if (OrgChart.xsl: Lr	n1:Shares*125.00

4. Start the Debugger. The results of the XPath you entered for the tracepoint appear in the Trace window.



```
<xsl:for-each select="n1:Person">
  <xsl:if test="n1:Shares &lt;= 0 or not (n1:Shares)">
    <xsl:for-each select="n1:First">
      <span style="color:#004040; font-family:Arial;
        <xsl:apply-templates />
      </span>
```

Trace

Element	Location	Result
xsl:if	OrgChart.xsl: Ln	xs:decimal(187500.0)
xsl:if	OrgChart.xsl: Ln	xs:decimal(0.0)
xsl:if	OrgChart.xsl: Ln	
xsl:if	OrgChart.xsl: Ln	xs:decimal(250000.0)
xsl:if	OrgChart.xsl: Ln	xs:decimal(125000.0)

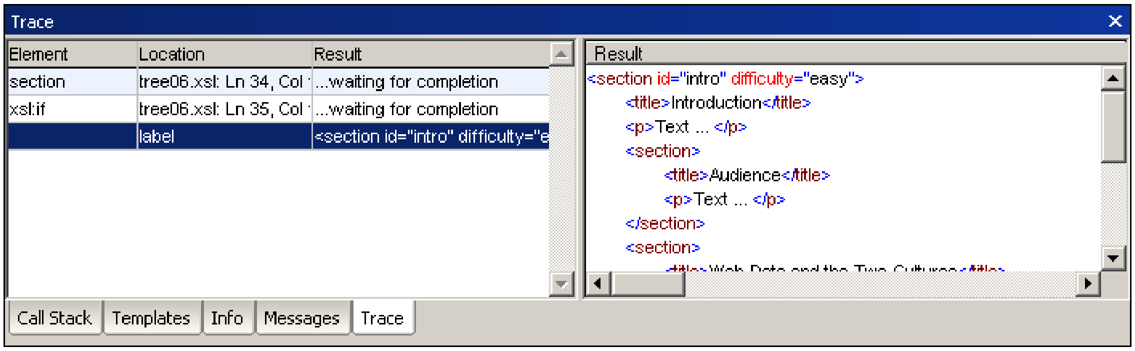
Result

xs:decimal(187500.0)

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.



Trace

Element	Location	Result
section	tree06.xsl: Ln 34, Col	...waiting for completion
xsl:if	tree06.xsl: Ln 35, Col	...waiting for completion
	label	<section id="intro" difficulty="e

Result

```
<section id="intro" difficulty="easy">
    <title>Introduction</title>
    <p>Text ... </p>
    <section>
        <title>Audience</title>
        <p>Text ... </p>
    </section>
    <section>
        <title>Web Data and the Two Cultures</title>
```

Disabling/enabling tracepoints

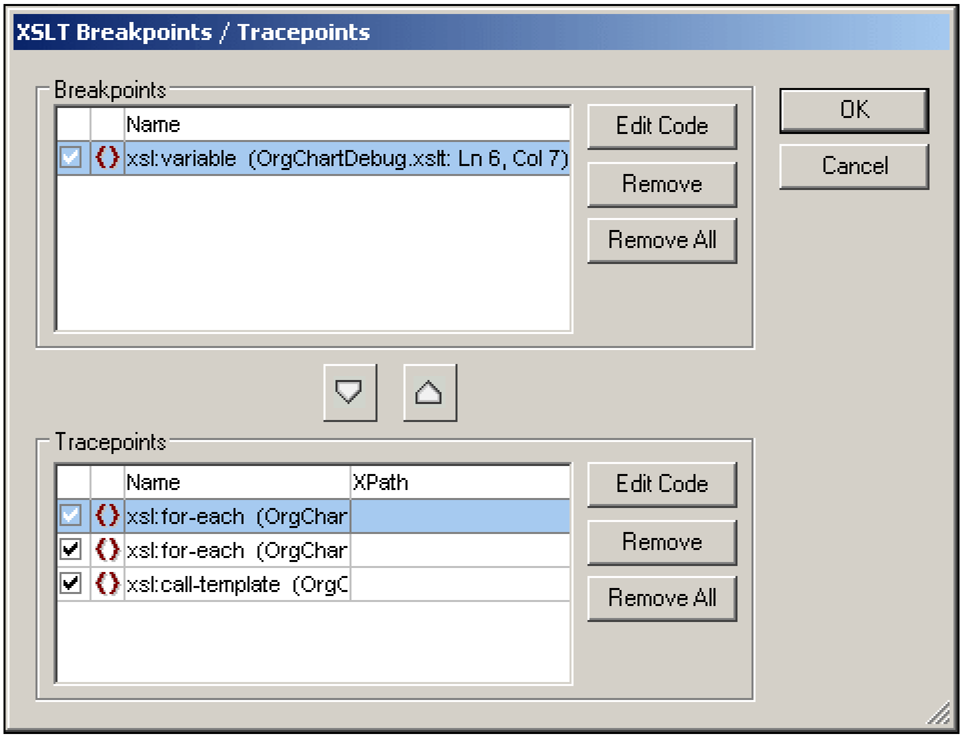
After inserting tracepoints, you can disable them if you wish to skip over them without having to delete them. You can enable them again when necessary.

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.



Breakpoints

		Name
☑	{}	xsl:variable (OrgChartDebug.xslt: Ln 6, Col 7)

Tracepoints

		Name	XPath
☑	{}	xsl:for-each (OrgChar	
☑	{}	xsl:for-each (OrgChar	
☑	{}	xsl:call-template (OrgC	

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.

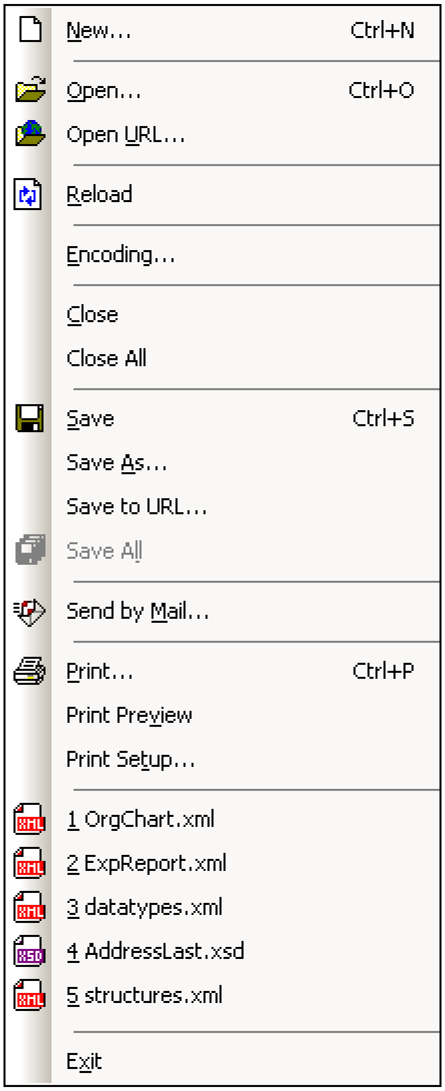
11 User Reference

The **User Reference** section contains a complete description of all XMLSpy menu commands and explains their use in general. We've tried to make this user manual as comprehensive as possible. 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 are still not able to have your problem satisfactorily addressed, please do not hesitate to contact us through the Support Center on the Altova website.

Note that in the File and Edit menus, all standard Windows commands are supported, as well as additional XML- and Internet-related commands.

11.1 File Menu

The **File** menu contains all commands relevant to manipulating files, in the order common to most Windows software products.

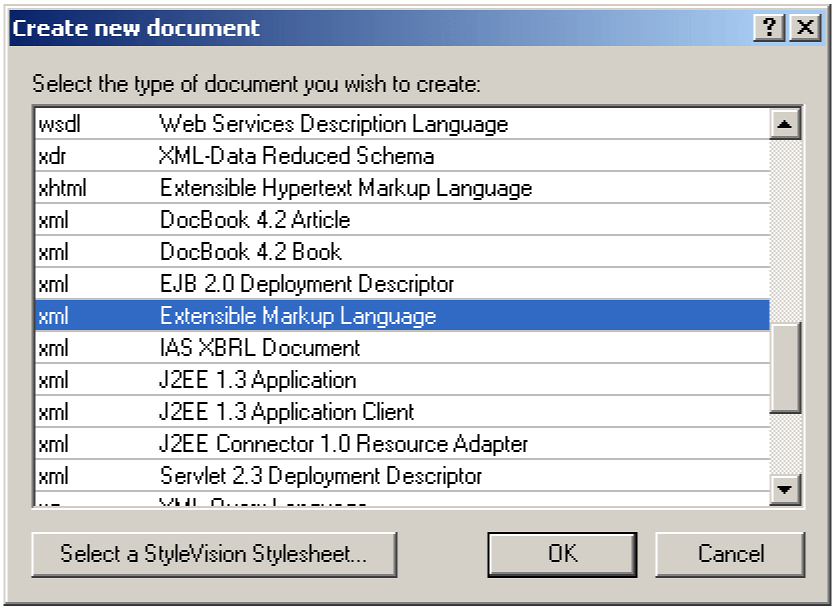


In addition to the standard New, Open, Save, Print, Print Setup, and Exit commands, XMLSpy offers a range of XML- and application-specific commands.

11.1.1 New...

Ctrl+N

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.



Select the type of document you wish to create:

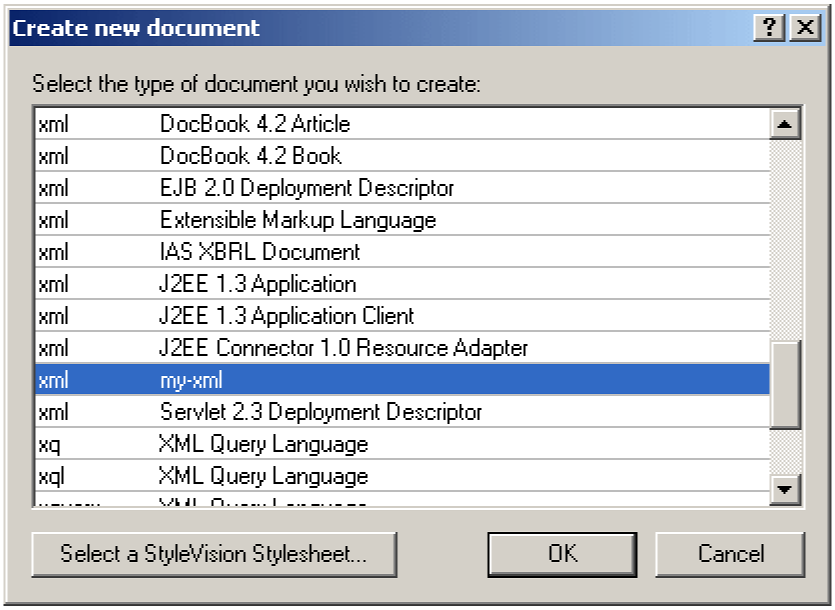
wsdl	Web Services Description Language
xdr	XML-Data Reduced Schema
xhtml	Extensible Hypertext Markup Language
xml	DocBook 4.2 Article
xml	DocBook 4.2 Book
xml	EJB 2.0 Deployment Descriptor
xml	Extensible Markup Language
xml	IAS XBRL Document
xml	J2EE 1.3 Application
xml	J2EE 1.3 Application Client
xml	J2EE Connector 1.0 Resource Adapter
xml	Servlet 2.3 Deployment Descriptor

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.



Create new document

Select the type of document you wish to create:

xml	DocBook 4.2 Article
xml	DocBook 4.2 Book
xml	EJB 2.0 Deployment Descriptor
xml	Extensible Markup Language
xml	IAS XBRL Document
xml	J2EE 1.3 Application
xml	J2EE 1.3 Application Client
xml	J2EE Connector 1.0 Resource Adapter
xml	my-xml
xml	Servlet 2.3 Deployment Descriptor
xq	XML Query Language
xql	XML Query Language

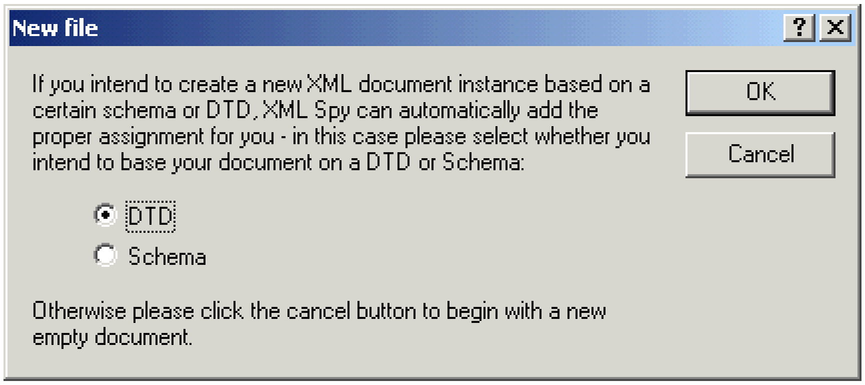
4. To open the template, select **File | New**, and then the template (`my-xml`, in this case).

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.



New file

If you intend to create a new XML document instance based on a certain schema or DTD, XML Spy can automatically add the proper assignment for you - in this case please select whether you intend to base your document on a DTD or Schema:

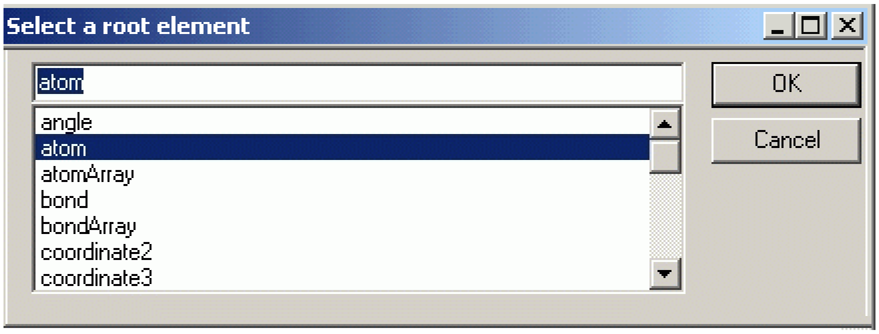
● DTD
○ Schema

Otherwise please click the cancel button to begin with a new empty document.

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.

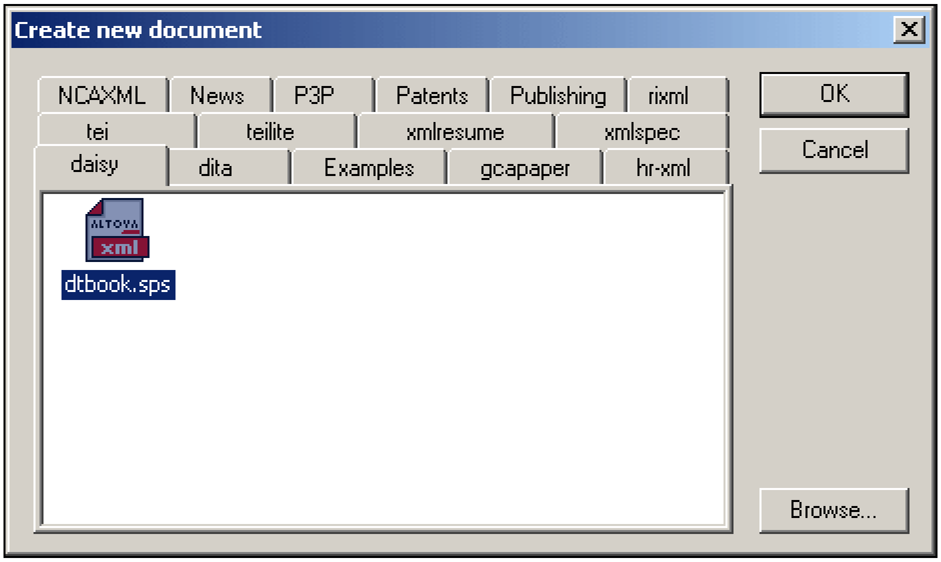
Specifying the document element of a new XML document

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.

11.1.2 Open...

Ctrl+O

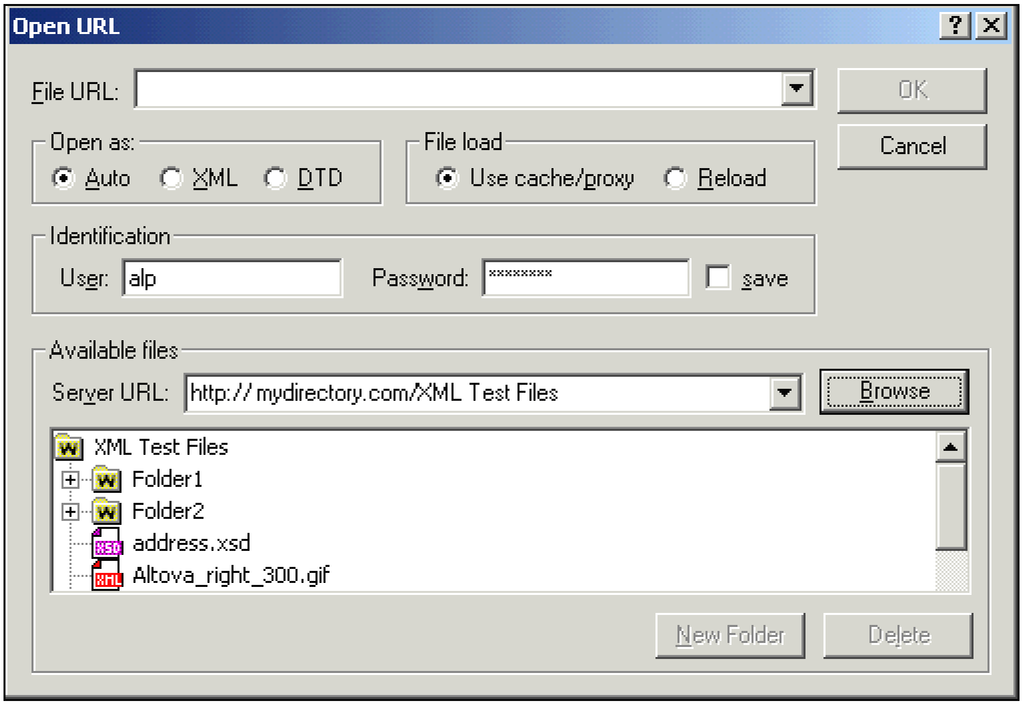
The **Open...** command pops up the familiar Windows Open dialog (*screenshot below*), and allows you to open any XML-related document or text document. In the Open dialog, you can select more than one file to open. Use the Files of Type combo box to restrict the kind of files displayed in the dialog 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 click Recheck 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 click Revalidate to revalidate.

11.1.3 Open URL...

The **Open URL...** command opens non-local files from a URL using http and WebDAV.

To open a URL:

1. Click the Open URL command. This opens the Open URL dialog.



Open as: ● Auto ○ XML ○ DTD

File load: ● Use cache/proxy ○ Reload

User: alp

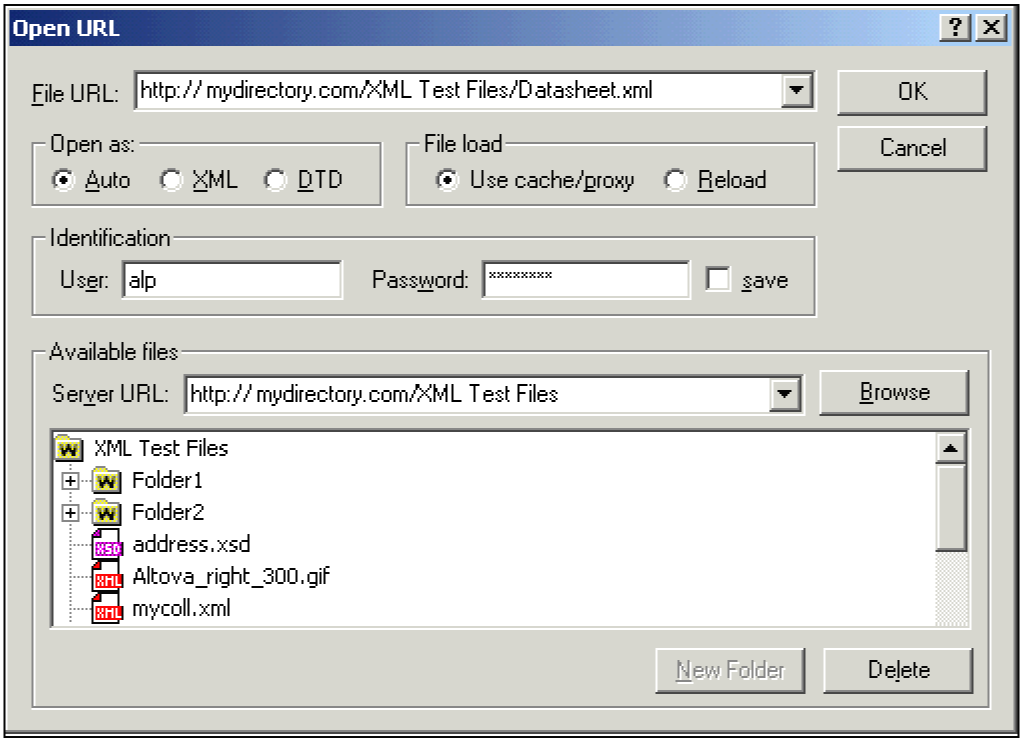
Password: *********

□ save

Server URL: http:// mydirectory.com/XML Test Files

XML Test Files
- Folder1
- Folder2
- address.xsd
- Altova_right_300.gif

2. Enter the URL you want to access, in the **Server URL** field.
3. Enter your User-ID in the **User** and **Password** fields, if the server is password protected.
4. Click **Browse** to view and navigate the directory structure of the server.
5. Click the file you want to load into XMLSpy.



Open URL

File URL: http:// mydirectory.com/XML Test Files/Datasheet.xml

Open as: ● Auto ○ XML ○ DTD

File load: ● Use cache/proxy ○ Reload

Identification

User: alp Password: xxxxxxxx □ save

Available files

Server URL: http:// mydirectory.com/XML Test Files

XML Test Files
- Folder1
- Folder2
- address.xsd
- Altova_right_300.gif
- mycoll.xml

The file URL appears in the File URL field. The **OK** button only becomes active at this point.

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

Please note: The Browse function is only available on servers which support the FTP, HTTP, and HTTPS (if the server supports WebDAV) protocols, and on servers that support WebDAV.

File load
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.

Repositories
Repositories supported by XMLSpy come in three flavors:

- Generic version control systems, such as Microsoft Visual Source-Safe (or compatible products)
- Generic web servers, such as FTP or WebDAV servers
- Specialized XML or schema repositories, such as Tamino, TEXTML Server, Virtuoso, or XML Canon.

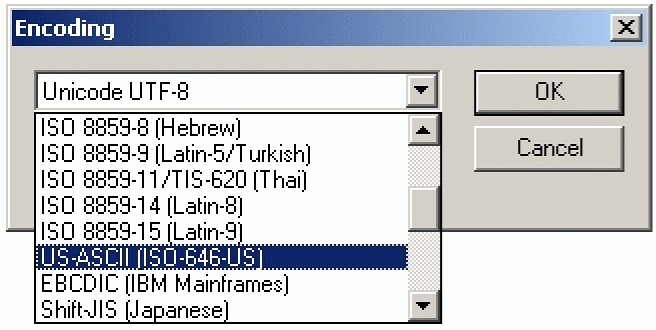
XMLSpy currently supports Source-Safe, FTP, and WebDAV servers, as well as Virtuoso through WebDAV. Other repository interfaces will be available in future versions.

11.1.4 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).

11.1.5 Encoding...

The **Encoding...** command lets you view the current encoding of a file and select a different encoding when saving the current document the next time.



If you select a different encoding than the one in use before, the encoding specification in the XML declaration in the prolog will be adjusted accordingly. For 16-bit and 32-bit per character encodings (UTF-16, UCS-2, and UCS-4) you can also specify the byte-order to be used for the file. You can also enter the new encoding into the encoding specification of the XML-declaration. When saving a document, XMLSpy automatically checks the encoding specification and opens a dialog box if it cannot recognize the encoding name entered by the user.

Please note: If your document contains characters that cannot be represented in the selected encoding, you will get a warning message as soon as you save your file.

11.1.6 Close

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.

11.1.7 Close All

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.

11.1.8 Save

Ctrl+S

The **Save** 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.

11.1.9 Save As...

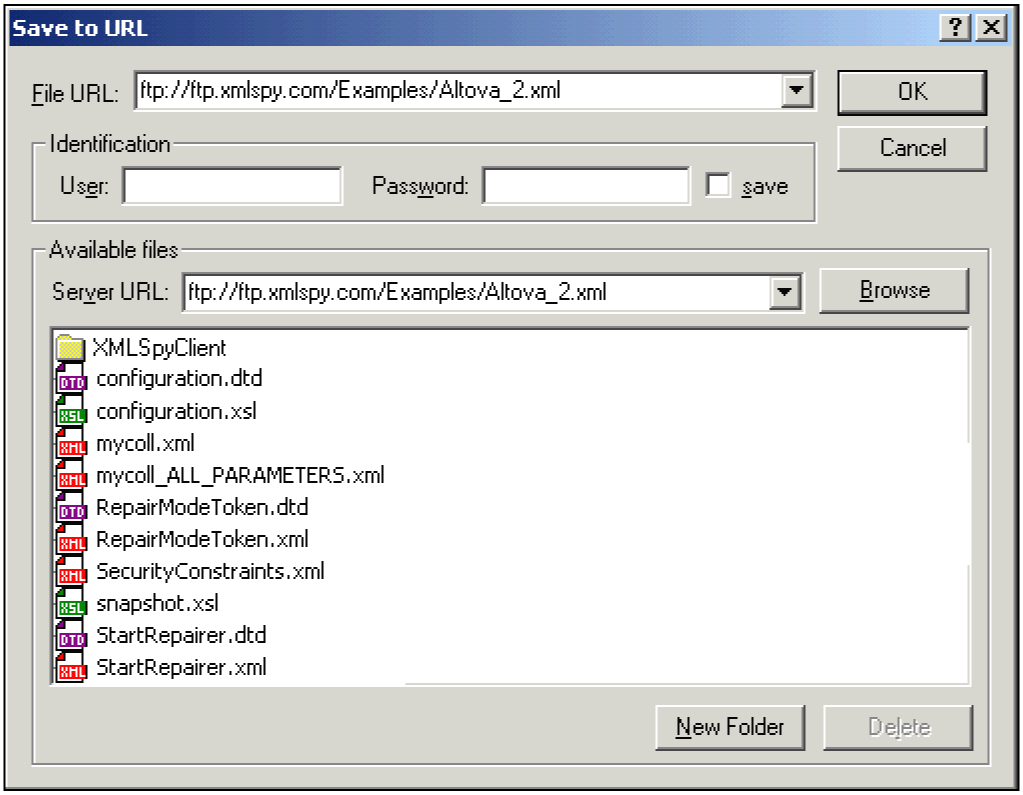
The **Save As...** command pops up the familiar Windows Save As dialog box, in which you enter the name and location of the file you wish to save the active file as. The same checks and validations occur as for the **Save** command.

11.1.10 Save to URL...

The **Save to URL...** command allows you to save files to a specified URL.

To save an XML document to a URL:

1. Make the file active in the Main Window.
2. Select the menu option **File | Save to URL** The following dialog appears.



File URL: ftp://ftp.xmlspy.com/Examples/Altova_2.xml

Server URL: ftp://ftp.xmlspy.com/Examples/Altova_2.xml

- XMLSpyClient
- configuration.dtd
- configuration.xsl
- mycoll.xml
- mycoll_ALL_PARAMETERS.xml
- RepairModeToken.dtd
- RepairModeToken.xml
- SecurityConstraints.xml
- snapshot.xsl
- StartRepairer.dtd
- StartRepairer.xml

3. Click the **Browse** button to see and navigate the directory structure of the server.
4. Enter the file name in the URL field or mark the file in the Available Files list box if you want to overwrite it.

Please note:

- The Browse function is only available on servers which support the FTP, HTTP, and HTTPS (if the server supports WebDAV) protocols, and on servers that support WebDAV.
- If the server you connect to is password protected, enter the required data and click the **OK** button to connect. You can also enter these data in the User and Password fields.

The Save check box, supplies the password data for the logon attempt.

Repositories
Repositories supported by XMLSpy come in three flavors:

- Generic version control systems, such as Microsoft Visual Source-Safe (or compatible products)
- Generic web servers, such as FTP or WebDAV servers; and
- Specialized XML or schema repositories, such as Tamino, TEXTML Server, Virtuoso, or XML Canon.

XMLSpy currently supports Source-Safe, FTP, and WebDAV servers, as well as Virtuoso through WebDAV. Other repository interfaces will be available in future versions.

11.1.11 Save All

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.

11.1.12 Send by Mail...

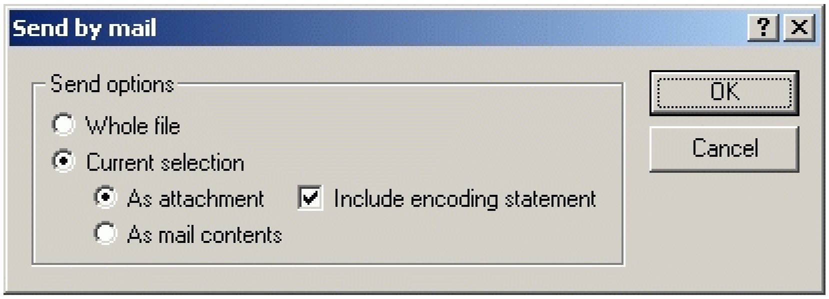
The **Send by Mail...** command lets you send XML document/s or selections from an XML document by e-mail. You can do any of the following:

- 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:

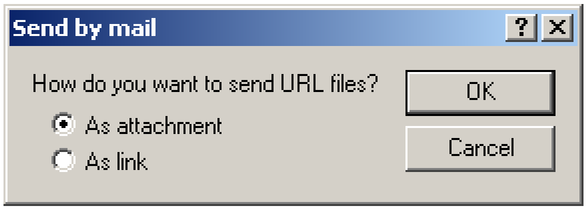
1. Make that document the active document in the Main Window. If you wish to send a selection or a group of files from a project, make the selection in the document or select the required files in the Project.
2. Click **Send by Mail...**. The following dialog opens:



Send options
○ Whole file
● Current selection
● As attachment ☑ Include encoding statement
○ As mail contents

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:



Send by mail

How do you want to send URL files?

● As attachment
○ As link

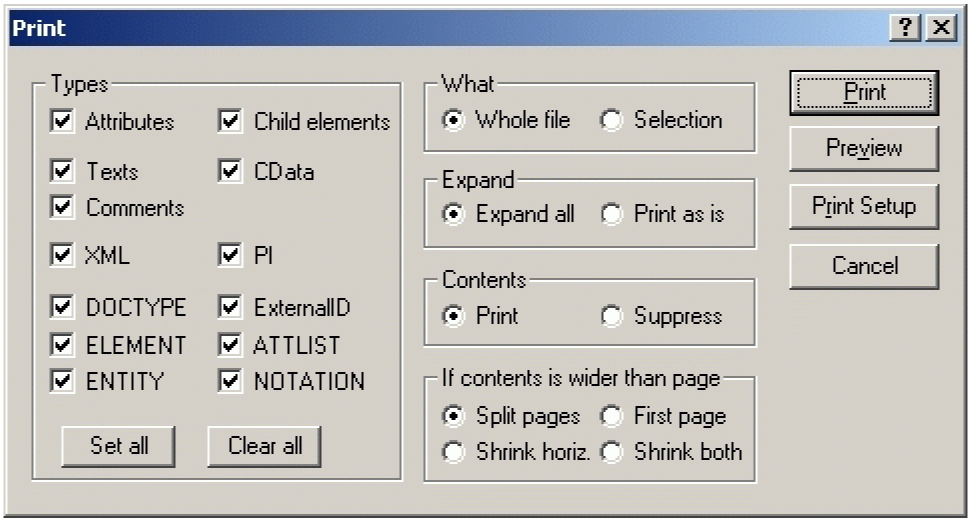
Select how the URL is to be sent and click **OK**.

11.1.13 Print...

Ctrl+P

The **Print...** command opens the Print dialog box, in which you can select printer options.

Clicking the **Print...** command in Enhanced 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.



Print

Types
☑ Attributes ☑ Child elements
☑ Texts ☑ CData
☑ Comments
☑ XML ☑ PI
☑ DOCTYPE ☑ ExternalID
☑ ELEMENT ☑ ATTLIST
☑ ENTITY ☑ NOTATION

What
● Whole file ○ Selection

Expand
● Expand all ○ Print as is

Contents
● Print ○ Suppress

If contents is wider than page
● Split pages ○ First page
○ Shrink horiz. ○ Shrink both

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.

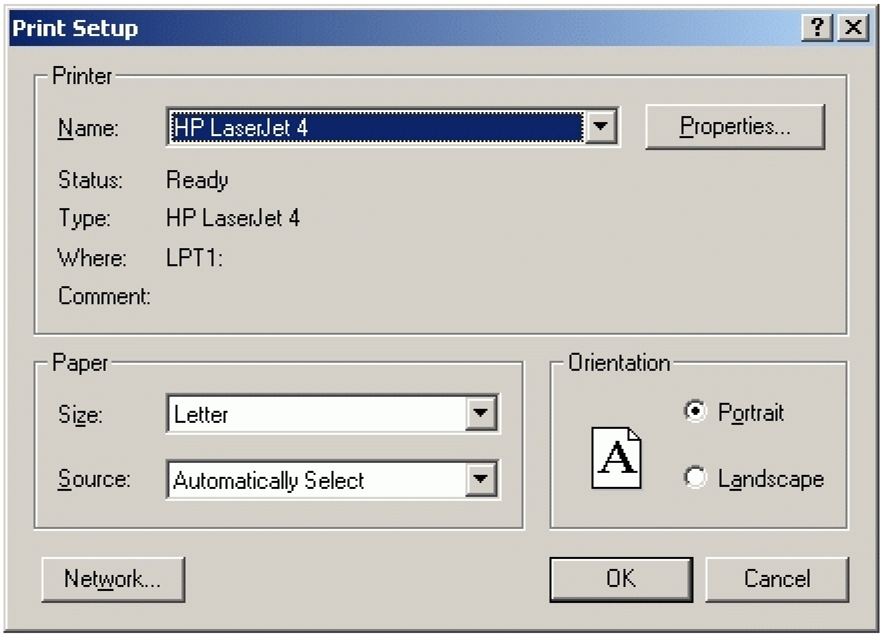
Please note: You can change column widths in Grid View to optimize the print output.

11.1.14 Print Preview

The **Print Preview** command opens the Print dialog box. Click the **Preview** button to display a print preview of the currently active document.

11.1.15 Print Setup...

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.



Name: HP LaserJet 4
Status: Ready
Type: HP LaserJet 4
Where: LPT1:
Comment:

Size: Letter
Source: Automatically Select

● Portrait
○ Landscape

The screenshot above shows the Print Setup dialog in which an HP LaserJet 4 printer attached to a parallel port (LPT1) is selected.

11.1.16 Most Recently Used Files

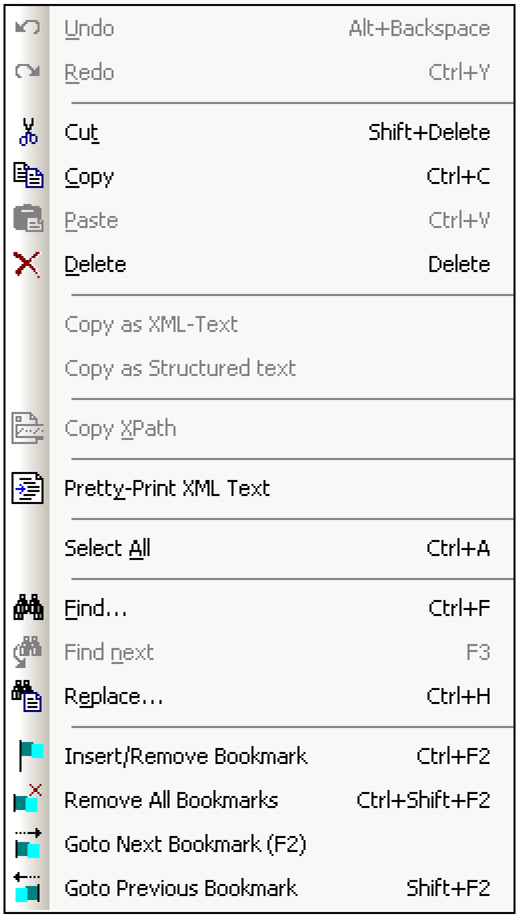
The **File** menu displays a list of the nine most recently used files, with the most recently opened file shown at the top of the list. You can open any of these files by clicking its name. To open a file in the list using the keyboard, press **ALT+F** to open the **File** menu, and then press the number of the file you want to open.

11.1.17 Exit

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.

11.2 Edit Menu

The **Edit** menu contains commands for editing documents in XMLSpy.



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.

11.2.1 Undo

Ctrl+Z

The **Undo** 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.

11.2.2 Redo

Ctrl+Y

The **Redo** 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.

11.2.3 Cut

Shift+Del or **Ctrl+X**

The **Cut** command copies the selected text or items to the clipboard and deletes them from their present location.

11.2.4 Copy

Ctrl+C

The **Copy** 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.

11.2.5 Paste

Ctrl+V

The **Paste** command inserts the contents of the clipboard at the current cursor position.

11.2.6 Delete

Del

The **Delete** command deletes the currently selected text or items without placing them in the clipboard.

11.2.7 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:

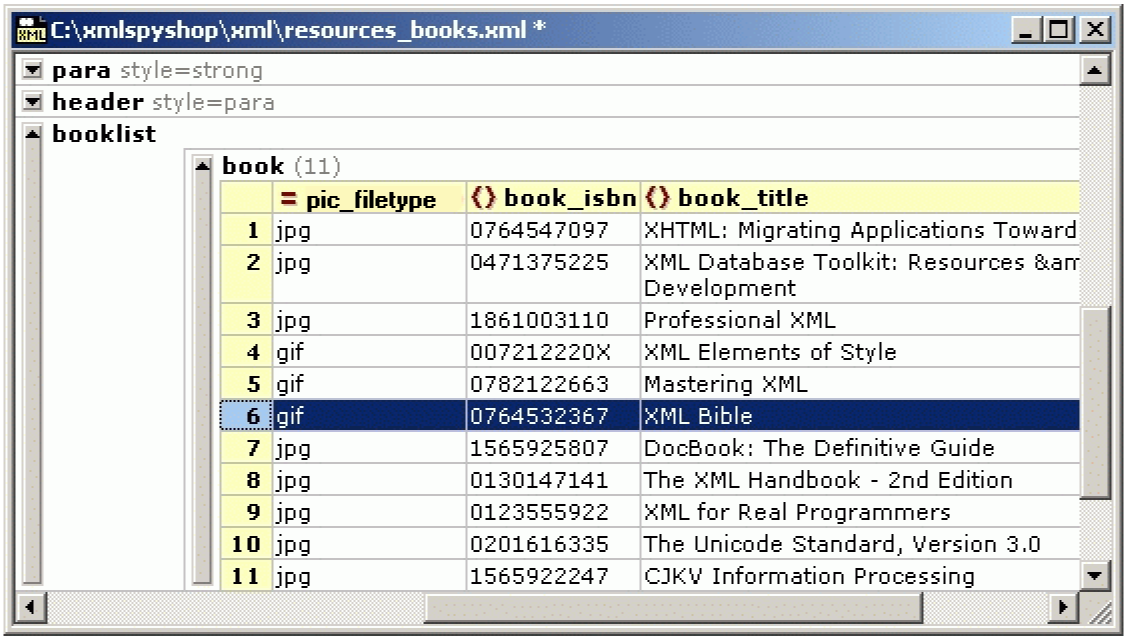
```
<row>
```

```
    <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**).

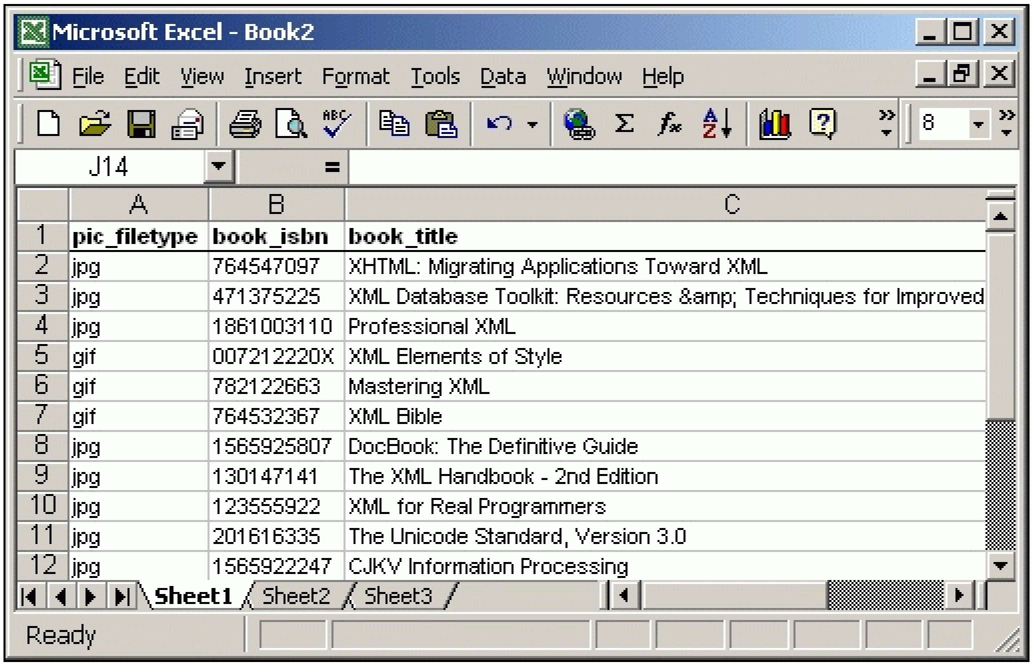
11.2.8 Copy as Structured Text

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).



	= pic_filetype	() book_isbn	() book_title
1	jpg	0764547097	XHTML: Migrating Applications Toward
2	jpg	0471375225	XML Database Toolkit: Resources &am 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:



	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 &amp; 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

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
```

11.2.9 Copy XPath

The **Copy XPath** command creates an XPath expression that selects the currently highlighted 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, for example:

- /company/person/last if the last element node is selected
- /company/person if one or more person element nodes are selected

The Copy XPath command is active only in Enhanced Grid View.

11.2.10 Pretty-Print XML Text

The **Pretty-Print XML Text** command reformats your XML document in Text View to give a structured display of the document. Each child node is offset from its parent by the amount of space specified in the Save File option of the File tab of the Options dialog (**Tools | Options**). Note that the XML document must be well-formed for this command to work.

11.2.11 Select All

Ctrl+A

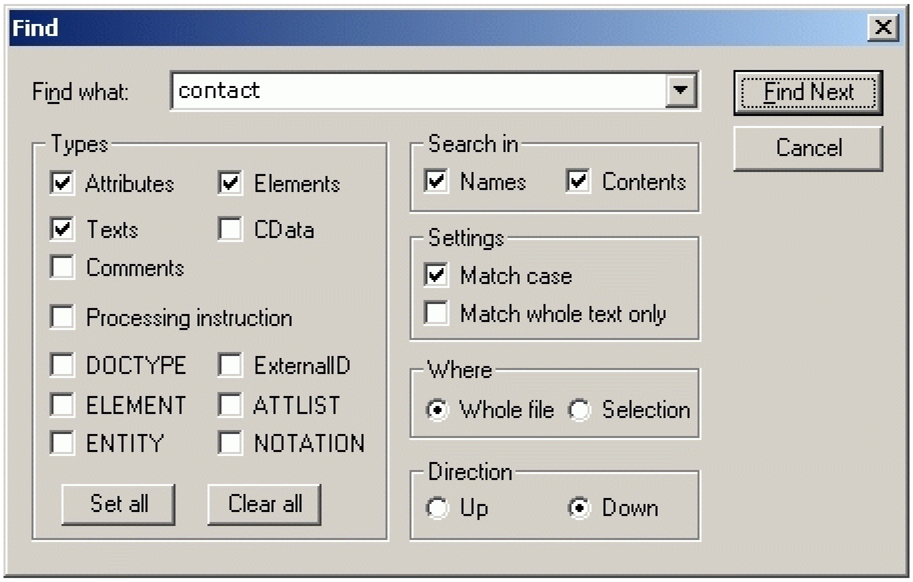
The **Select All** command selects the contents of the entire document.

11.2.12 Find...

Ctrl+F

The **Find** command 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.



Find

Find what: contact

Types
☑ Attributes ☑ Elements
☑ Texts ☐ CData
☐ Comments
☐ Processing instruction
☐ DOCTYPE ☐ ExternalID
☐ ELEMENT ☐ ATTLIST
☐ ENTITY ☐ NOTATION

Search in
☑ Names ☑ Contents

Settings
☑ Match case
☐ Match whole text only

Where
● Whole file ○ Selection

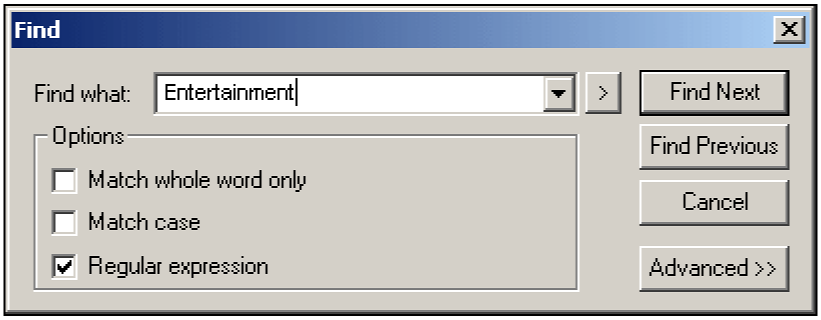
Direction
○ Up ● Down

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.



Find

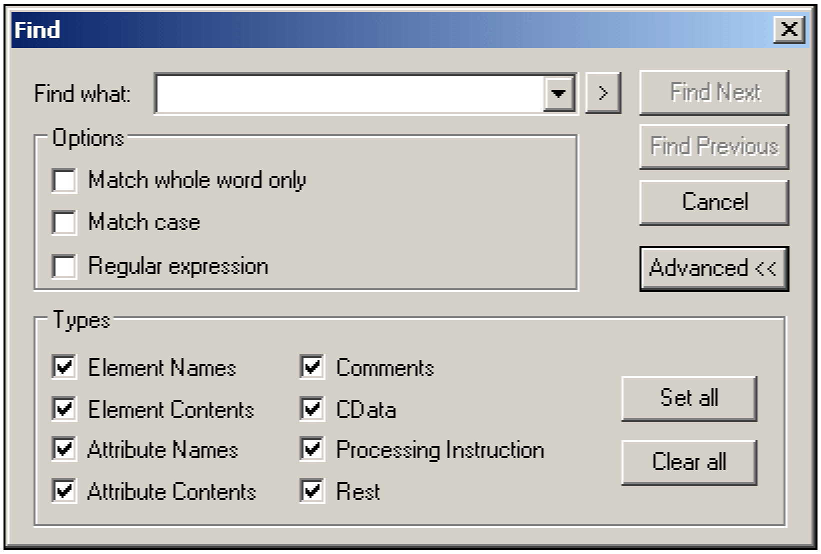
Find what: Entertainment

Options
☐ Match whole word only
☐ Match case
☑ Regular expression

The following Find options are available:

- Match whole word only: Only the exact words in the text will be matched.
- 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.



Options

□ Match whole word only

□ Match case

□ Regular expression

Types

☑ Element Names ☑ Comments

☑ Element Contents ☑ CData

☑ Attribute Names ☑ Processing Instruction

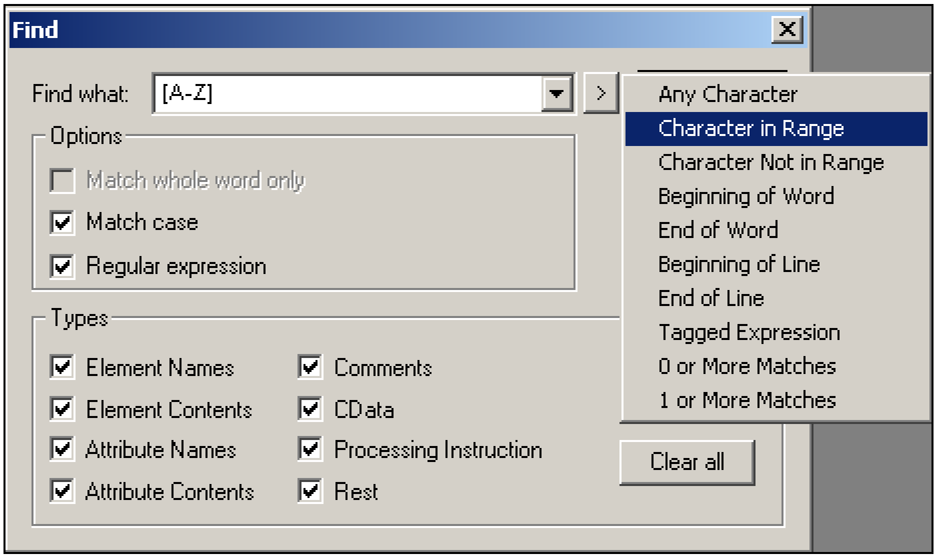
☑ Attribute Contents ☑ Rest

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.



Find what: [A-Z]

Options

□ Match whole word only

☑ Match case

☑ Regular expression

Types

☑ Element Names ☑ Comments

☑ Element Contents ☑ CData

☑ Attribute Names ☑ Processing Instruction

☑ Attribute Contents ☑ Rest

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.

11.2.13 Find next

F3

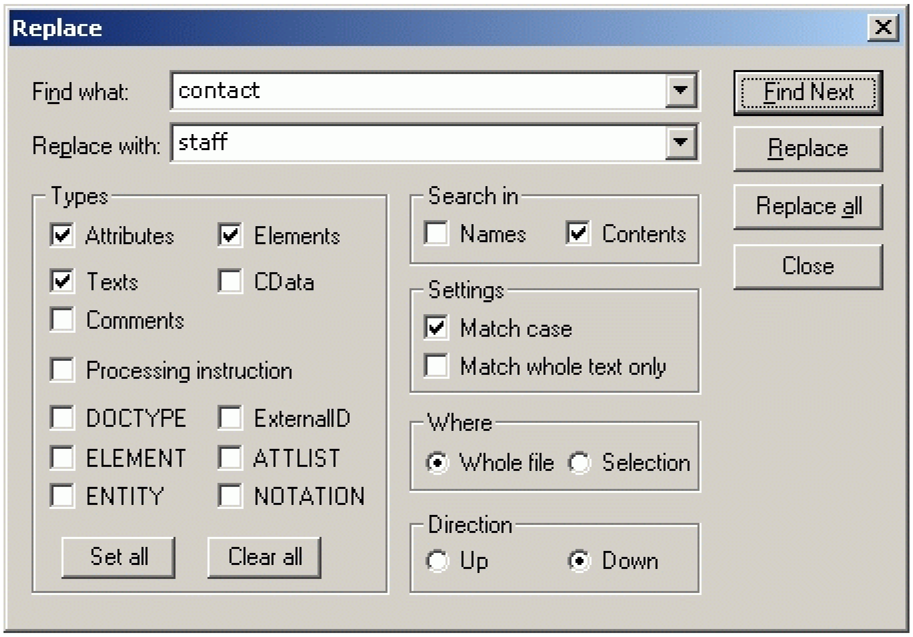
The **Find next** command repeats the last Find command to search for the next occurrence of the requested text.

11.2.14 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.



Find what: contact

Replace with: staff

Types
☑ Attributes ☑ Elements
☑ Texts □ CData
□ Comments
□ Processing instruction
□ DOCTYPE □ ExternalID
□ ELEMENT □ ATTLIST
□ ENTITY □ NOTATION

Search in
□ Names ☑ Contents

Settings
☑ Match case
□ Match whole text only

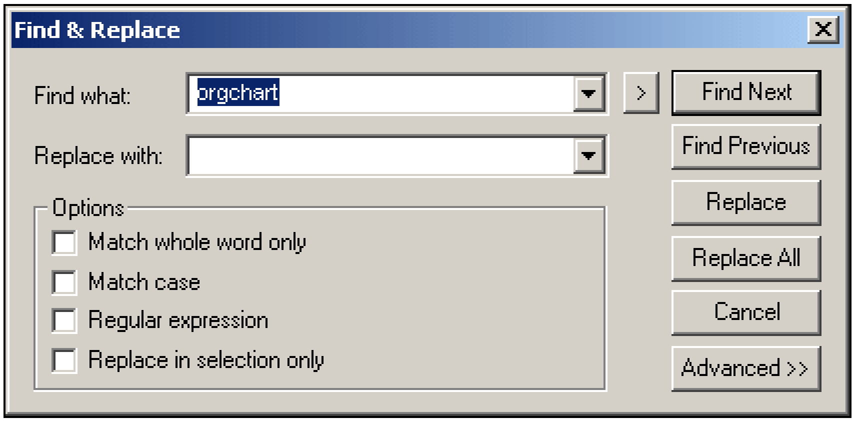
Where
● Whole file ○ Selection

Direction
○ Up ● Down

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.

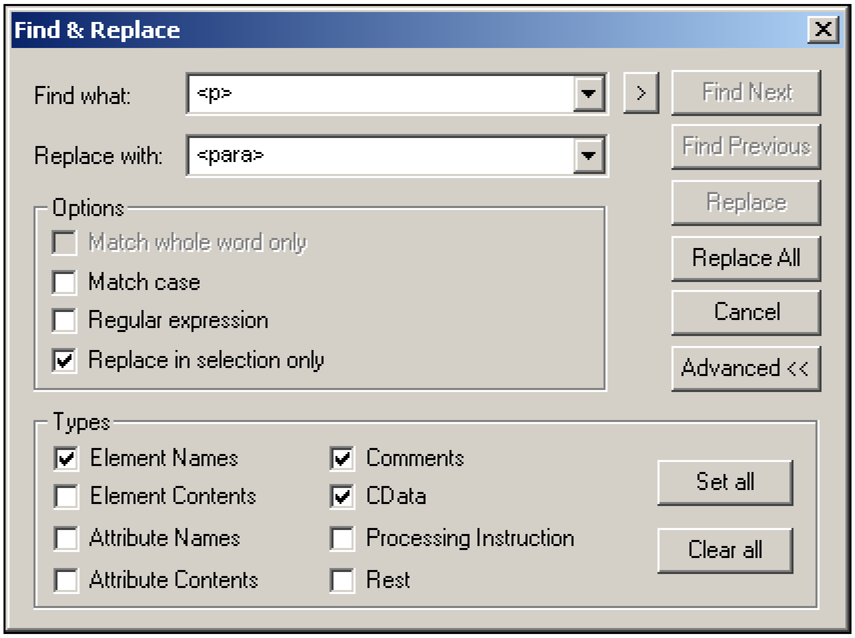


Find what: orgchart

Replace with:

Options
□ Match whole word only
□ Match case
□ Regular expression
□ Replace in selection only

Clicking the **Advanced** button opens the Types options (see Find... for details).



Find what: <p>

Replace with: <para>

Options

- □ Match whole word only
- □ Match case
- □ Regular expression
- ☑ Replace in selection only

Types

- ☑ Element Names
- □ Element Contents
- □ Attribute Names
- □ Attribute Contents
- ☑ Comments
- ☑ CData
- □ Processing Instruction
- □ Rest

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.

11.2.15 Insert/Remove Bookmark

Ctrl+F2

The **Insert/Remove Bookmark** command 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.

11.2.16 Remove All Bookmarks

Ctrl+Shift+F2

The **Remove All Bookmarks** command removes all the currently defined bookmarks. This command is only available in Text View.

Please note: The Undo command does not undo the effects of this command.

11.2.17 Goto Next Bookmark

F2

The **Goto Next Bookmark** command places the text cursor at the beginning of the next bookmarked line. This command is only available in the Text View.

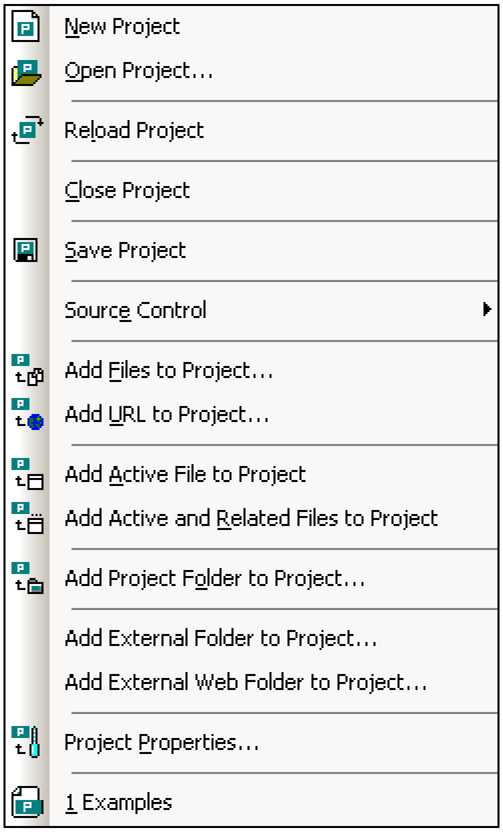
11.2.18 Goto Previous Bookmark

Shift+F2

The **Goto Previous Bookmark** command places the text cursor at the beginning of the previous bookmarked line. This command is only available in the Text View.

11.3 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.


New Project
Open Project...
Reload Project
Close Project
Save Project
Source Control
Add Files to Project...
Add URL to Project...
Add Active File to Project
Add Active and Related Files to Project
Add Project Folder to Project...
Add External Folder to Project...
Add External Web Folder to Project...
Project Properties...
1 Examples

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.

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.

11.3.1 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.

11.3.2 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.

11.3.3 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.

11.3.4 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's title bar.

11.3.5 Save Project

The **Save Project** command **saves** the current project. You can also save a project by making the project window active and clicking the 💾 icon.

11.3.6 Source control

XMLSpy supports Microsoft Source-Safe and other compatible repositories.

Open source revision control systems:

- Concurrent Versions System (CVS).
  The Jalindi Igloo software must be downloaded for XMLSpy to connect to CVS. Download this software component from the Altova components web site, or from the Jalindi.com site.

Commercial revision control systems:

- Microsoft Visual Source-Safe 6.0c
- PVCS Version Manager 6.7 *
- StarTeam 5.1 *

* Compatibility with this third-party tool depends on its implementation of SCC interface and cannot be expressly guaranteed.

A source control project is, however, not the same as a XMLSpy project. Source control projects are extremely directory dependent, whereas XMLSpy projects are logical constructions without direct directory dependence.

In the following description, Microsoft Source-Safe is used as the Source Control provider.

Please note:
Microsoft has defined a Registry Entry, where all SCC compatible programs can register themselves. XMLSpy only reads this entry.

HKEY_LOCAL_MACHINE\SOFTWARE\SourceCodeControlProvider\InstalledSCCProviders

Delta V
Extends the WebDAV protocol, where WebDAV is itself an extension of the HTTP 1.1 protocol. XMLSpy now supports Delta V at the project, folder and single file level via the context menu

(right click on the respective object). The Delta V protocol is supported by XMLSpy but is only enabled, if the server capabilities on which the files under source control are placed, allow this.

Accessing files placed under source control on a Delta V enabled web server, automatically enables the Delta V versioning commands: check in/out, undo check out.

Delta V commands can be accessed by right-clicking:

- the file tab of an open file in the main window
- a file in the Open URL browse window
- a file in an external web folder of a project

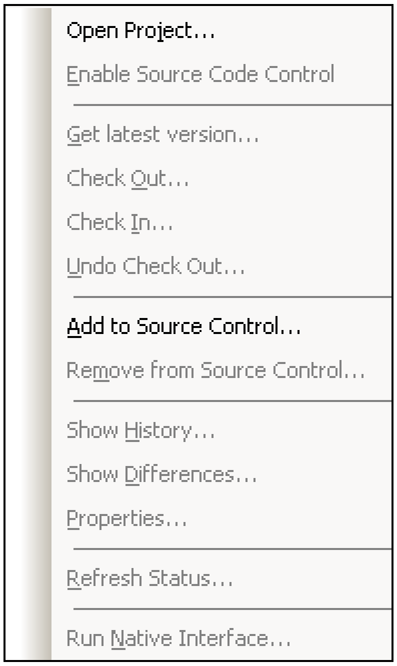
Delta V is the **versioning** part of the WebDAV standard and supports:

- Check out
- Undo check out
- Check in (plus comment)

WebDAV capabilities include:

- Properties: list, add and remove
- Namespace operations: move and copy
- Collections:hierarchy operations and mkcol.

For more in-depth information on Delta V please see www.webdav.org/deltav.

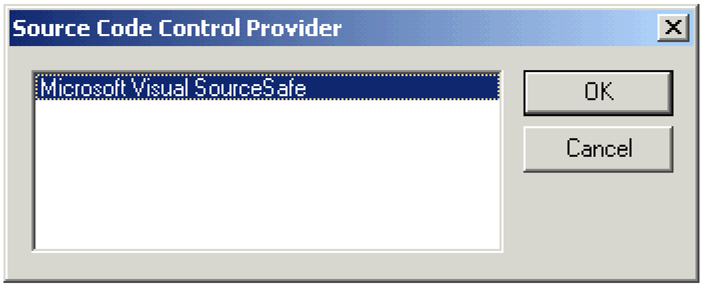


Open Project

The Open Project command mirrors (or creates) an existing source control project locally. This command can only take effect if an XMLSpy project has previously been added to source control provider using the menu option Add to Source Control.

1. Select **Project | Source Control | Open Project**.
2. Select a source control provider from the list of those installed on your PC, and confirm

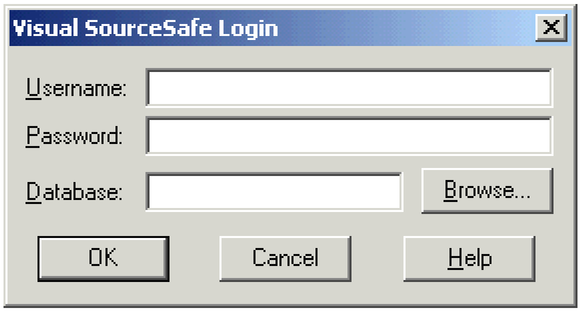
with **OK**.



Source Code Control Provider

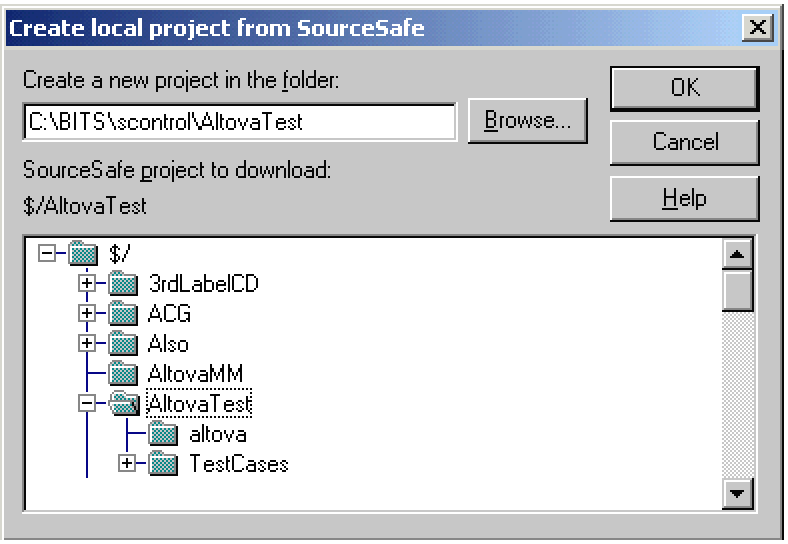
Microsoft Visual SourceSafe

3. Enter your login data in the Source Control Login dialog box, and confirm with **OK**. The Create local project from SourceSafe dialog appears.



Visual SourceSafe Login

4. Define the directory to contain the local project.
5. Select the Source control project you want to download.
   If the folder you define does not exist at the location, a dialog box opens prompting you to create it there. Click **Yes** to confirm the new directory.
6. Click **OK** to download the source control project.

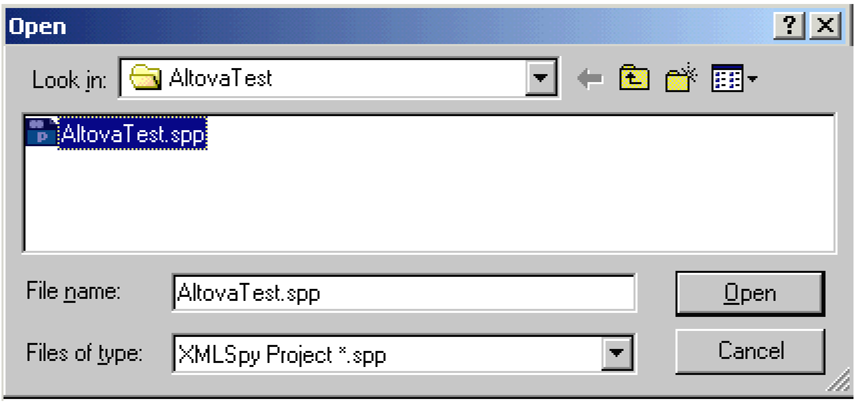


Create local project from SourceSafe

Create a new project in the folder: C:\BITS\scontrol\AltovaTest

SourceSafe project to download: $/AltovaTest

7. This opens the Open dialog box, and attempts to find the XMLSpy project file (*.spp) in the local directory. Click the project file you want to create, if there are several to choose from.



Look in: AltovaTest

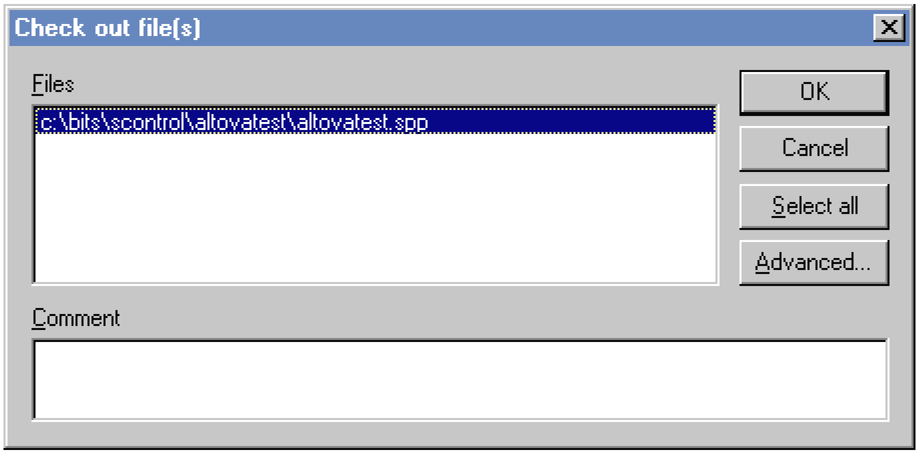
AltovaTest.spp

File name: AltovaTest.spp

Files of type: XMLSpy Project *.spp

A message box might appear at this point stating that the directory is under source control, offering you the option of checking it out or not.

8. Click **OK** if you want source control to be enabled.
   The Check out file(s) dialog box opens, allowing you to check out the *.spp file.
9. Add any comments to the file in the Comment text box and click **OK**, to check out the XMLSpy project file. Depending on your source control provider, the dialog box might contain extra command buttons. In this case the **Advanced...** button opens the Advanced Check Out Options dialog box.

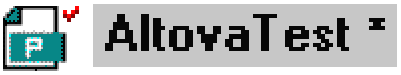


Files: c:\bits\scontrol\altovatest\altovatest.spp

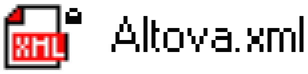
The *.spp file is checked out, and all the XML files contained in the XMLSpy project, are copied to the local directory.

Result in XMLSpy:
The AltovaTest directory has been created locally, and all files included in the XMLSpy project are made available in the project window. You can now work with these files as you normally do.

Source control symbols:

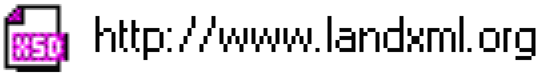


The **red check mark** denotes **checked out**, i.e. the AltovaTest project file has been checked out.
The asterisk denotes that changes have been made to the file, and you will be prompted to save it when you exit.



Altova.xml

The lock symbol, at the top right corner the XML symbol, denotes that the file is **under source control**, but is currently not checked out.



http://www.landxml.org

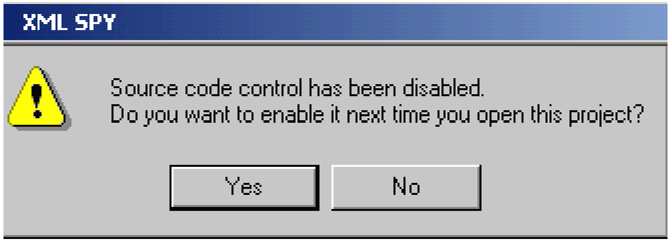
This URL document is part of the AltovaTest project but is **not under source control**. Documents opened via URL, cannot be placed under source control.

Enable Source Code Control

This command allows you to enable or disable source control for a project.

To disable source control for a project:

1. Click on the project file or any other file in the project window.
2. Select the menu option **Project | Source control** and deactivate the **Enable Source Code Control** check box.



XML SPY

Source code control has been disabled.
Do you want to enable it next time you open this project?

To **provisionally** disable source control for the project select **Yes**.

To **permanently** disable source control for the project select **No**.

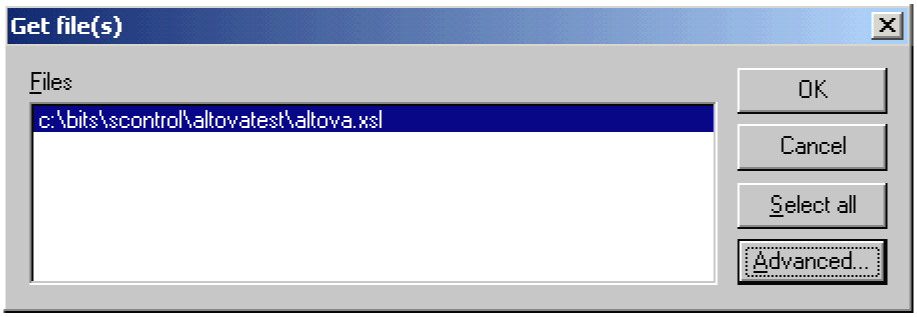
Get Latest Version

This command **gets** the latest source control version of the file you select.

To get the latest version of a file:

- Select **Project** | **Source Control** | **Get Latest Version**.

The file is a copy of the source control file and is placed in your local working directory.



Get file(s)

Files

c:\bits\scontrol\altovatest\altova.xsl

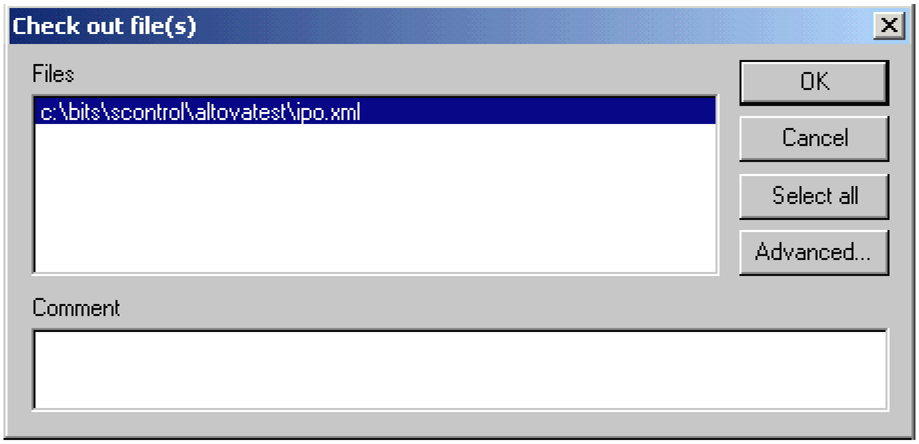
Check Out

This command **checks out** the latest source control version of file(s) you select, and flags them as "checked out" for all other users.

To check out files:

- Select **Project | Source Control | Check Out**.

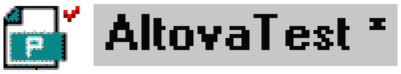
Shortcut: Right-click an item in the project window, and select "Check out" from the context menu.



c:\bits\scontrol\altovatest\ipo.xml

The following items can be checked out:

- Single files, click on the respective files (CTRL + click, for several)
- XMLSpy project folders, click on the folders (CTRL + click, for several)
- XMLSpy project, click on the project file icon.



The red check mark denotes that the file has been checked out.

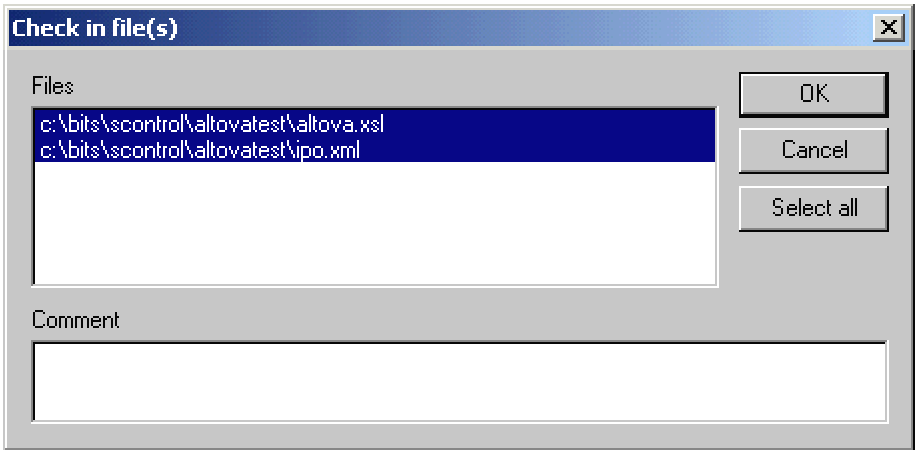
Check In

This command **checks in** the previously checked out files, i.e. your locally updated files, and places them in the source control project.

To check in files:

- Select **Project | Source Control | Check In**.

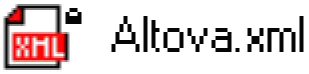
Shortcut: Right-click a checked out item in the project window, and select "Check in" from the Context menu.



c:\bits\scontrol\altovatest\altova.xsl
c:\bits\scontrol\altovatest\ipo.xml

The following items can be checked in:

- Single files, click on the respective files (CTRL + click, for several)
- XMLSpy project folders, click on the folders (CTRL + click, for several)
- A complete XMLSpy project, click on the project file icon.

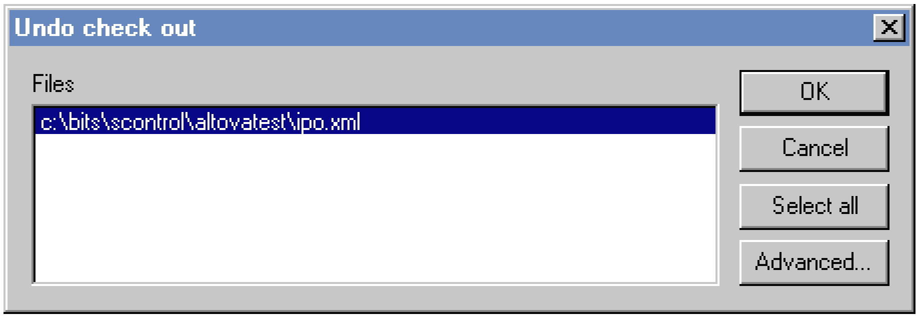


The lock symbol denotes that the file is **under source control**, but is currently not checked out.

Undo Check Out...

This command **rejects changes** made to previously checked out files, i.e. your locally updated files, and retains the old files in the source control project.

Shortcut: Right click a checked out item in the project window, and select **Undo Check out** from the Context menu.



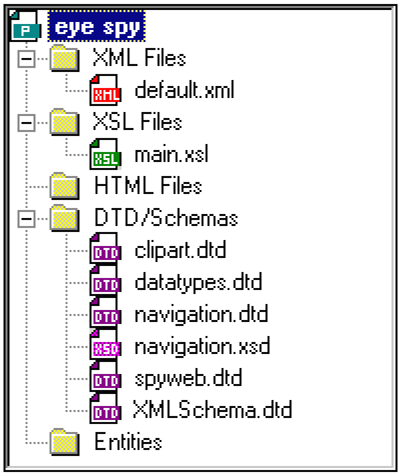
c:\bits\scontrol\altovatest\ipo.xml

The Undo check out option can apply to the following items:

- Single files, click on the respective files (CTRL + click, for several)
- XMLSpy project folders, click on the folders (CTRL + click, for several)
- XMLSpy project, click on the project file icon.

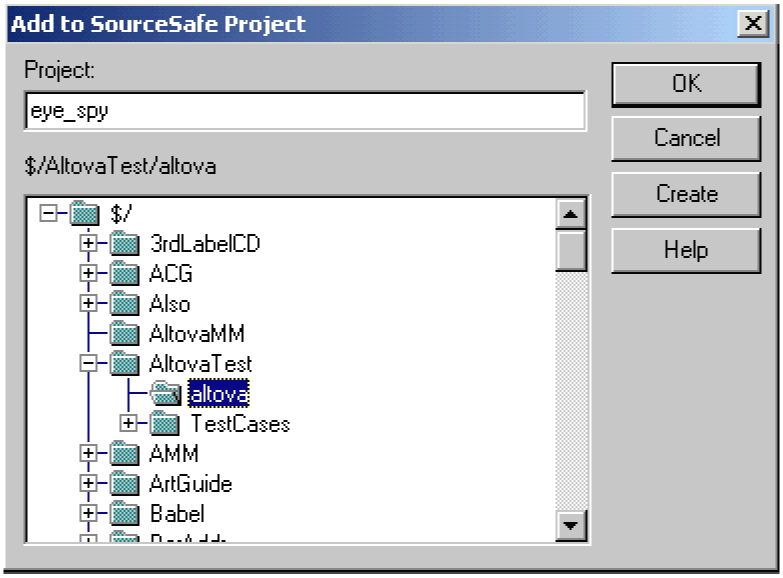
Add to Source Control

This command **adds** an XMLSpy project to your Source Control provider.



To add an XML project to the source control provider:

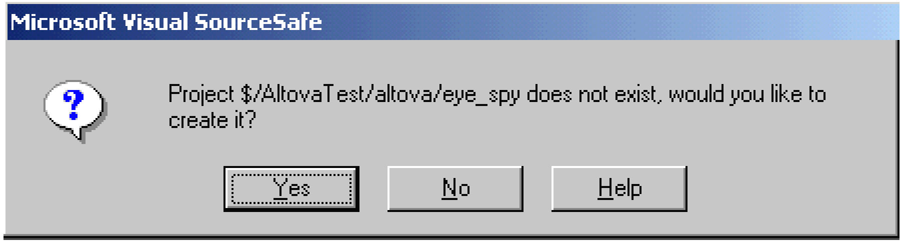
1. Click the XMLSpy project icon and select the menu option **Project | Source control | Add to Source control.**
2. Select your source control provider from the list box.
3. Enter the source control login data.
4. Select the source control project (directory) to which your XML project should be added, and confirm with **OK**.


Add to SourceSafe Project

Project: eye_spy

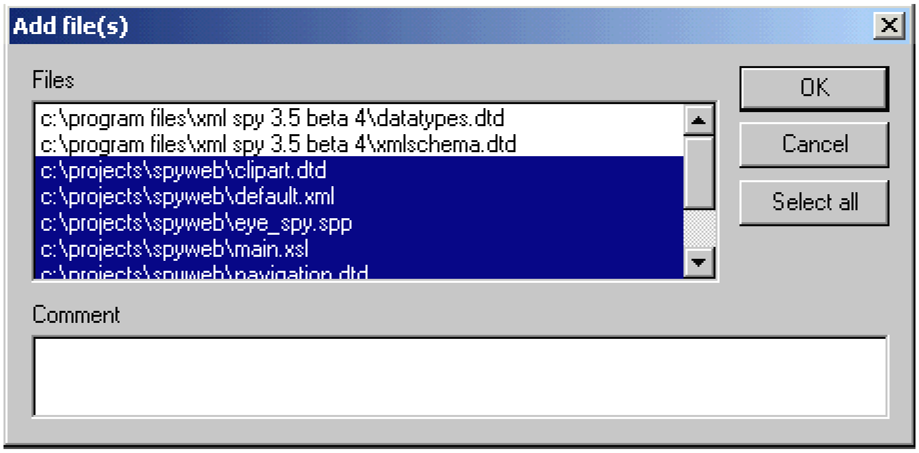
$/AltovaTest/altova

If the project does not exist, the following dialog box opens, prompting for more information.



Project $/AltovaTest/altova/eye_spy does not exist, would you like to create it?

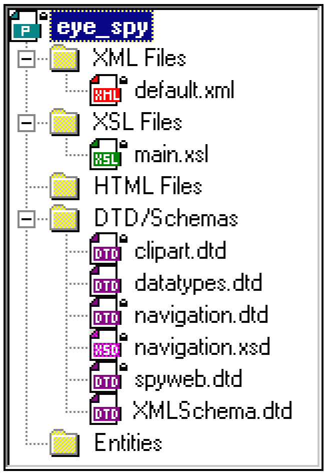
5. Select **Yes** to create the new project or **No** to cancel.



c:\program files\xml spy 3.5 beta 4\datatypes.dtd
c:\program files\xml spy 3.5 beta 4\xmlschema.dtd
c:\projects\spyweb\clipart.dtd
c:\projects\spyweb\default.xml
c:\projects\spyweb\eye_spy.spp
c:\projects\spyweb\main.xsl
c:\projects\spyweb\navigation.dtd

6. Select the XMLSpy project files that you want to be placed under control of the source control provider, and confirm with **OK**.

The files are transferred to the source control provider and are now under source control. These files are marked with the "lock symbol" in the project window. The files you did not transfer do not have the lock symbol (`datatypes.dtd` and `XMLschema.dtd` in this example).

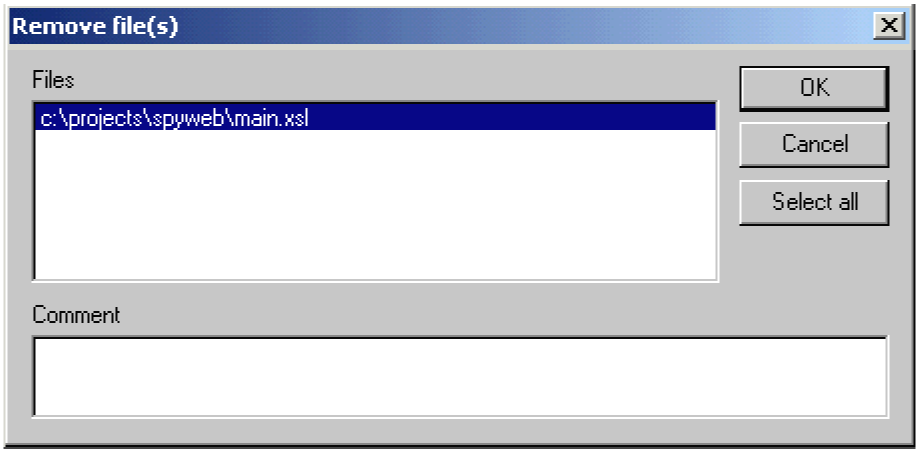


Remove from Source Control

This command **removes** previously added files from the source control provider.

To remove files from the source control provider:

- Select **Project** | **Source Control** | **Remove from Source Control**.



c:\projects\spyweb\main.xsl

The following items can be removed from source control:

- Single files, click on the respective files (CTRL + click, for several)
- XMLSpy project folders, click on the folders (CTRL + click for several)
- A complete XMLSpy project, click on the project file icon.

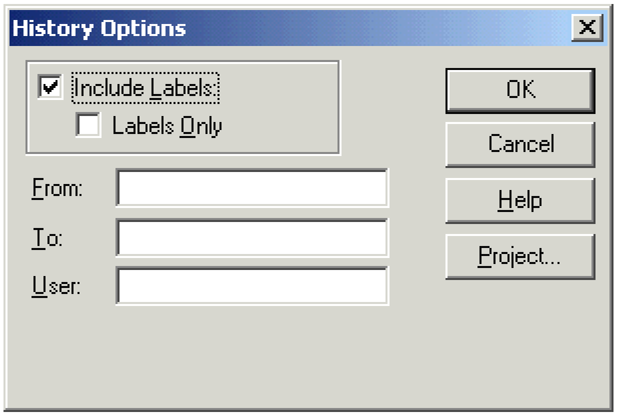
Show History

This command **displays** the history of a file. It can only be used on single files.

To show the history of a file:

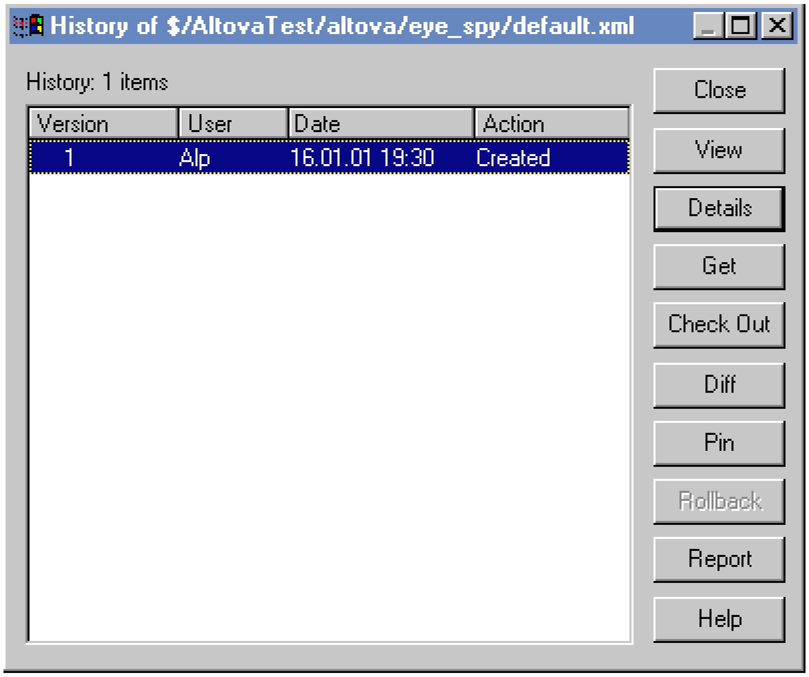
1. Click on the file in the project window.
2. Select the menu option **Project | Source control | Show history**.

   A dialog box prompting for more information may appear at this time (this example uses MS Source-Safe).

   ☑ Include Labels
   □ Labels Only
   From:
   To:
   User:

3. Select the appropriate entries and confirm with **OK**.



History of $/AltovaTest/altova/eye_spy/default.xml

History: 1 items

Version	User	Date	Action
1	Alp	16.01.01 19:30	Created

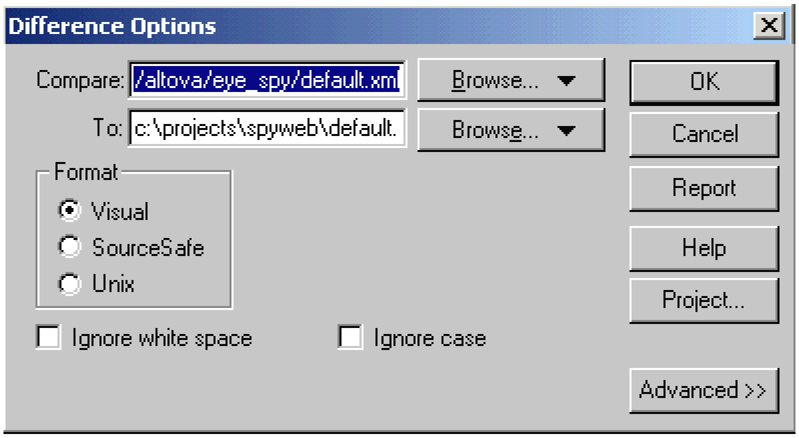
Show Differences

This command **displays** the differences between the file currently in the source control repository, and the file of the same name that you have checked out.

This command can only be used on single files.

To show the differences between two files:

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.



Difference Options

Compare: /altova/eye_spy/default.xml

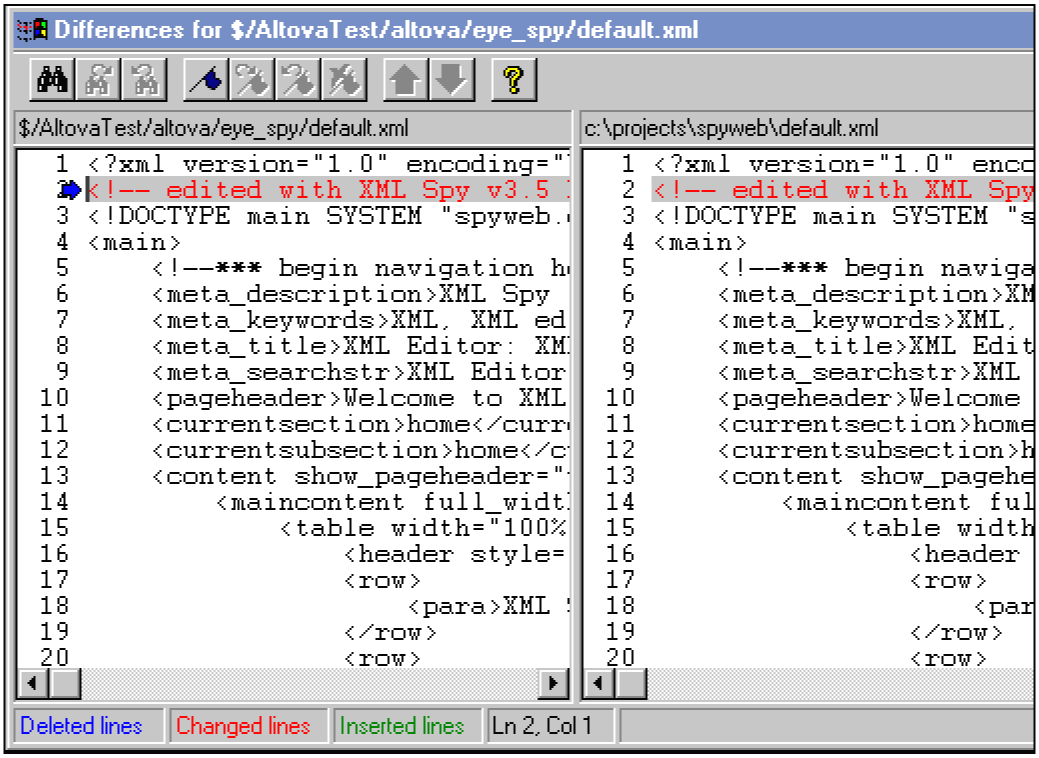
To: c:\projects\spyweb\default.

Format

● Visual
○ SourceSafe
○ Unix

□ Ignore white space   □ Ignore case

3. Select the appropriate entries and confirm with **OK**.


Differences for $/AltovaTest/altova/eye_spy/default.xml
$/AltovaTest/altova/eye_spy/default.xml
c:\projects\spyweb\default.xml
Deleted lines
Changed lines
Inserted lines
Ln 2, Col 1

```
<?xml version="1.0" encoding="
<!-- edited with XML Spy v3.5 
<!DOCTYPE main SYSTEM "spyweb.
<main>
    <!--*** begin navigation h
    <meta_description>XML Spy 
    <meta_keywords>XML, XML ed
    <meta_title>XML Editor: XM
    <meta_searchstr>XML Editor
    <pageheader>Welcome to XML
    <currentsection>home</curr
    <currentsubsection>home</c
    <content show_pageheader="
        <maincontent full_widt
            <table width="100%
                <header style=
                <row>
                    <para>XML 
                </row>
                <row>
```

```
<?xml version="1.0" enco
<!-- edited with XML Spy
<!DOCTYPE main SYSTEM "s
<main>
    <!--*** begin naviga
    <meta_description>XM
    <meta_keywords>XML,
    <meta_title>XML Edit
    <meta_searchstr>XML
    <pageheader>Welcome
    <currentsection>home
    <currentsubsection>h
    <content show_pagehe
        <maincontent ful
            <table width
                <header
                <row>
                    <par
                </row>
                <row>
```

The differences between the two files are highlighted in both windows (this example uses MS Source-Safe).

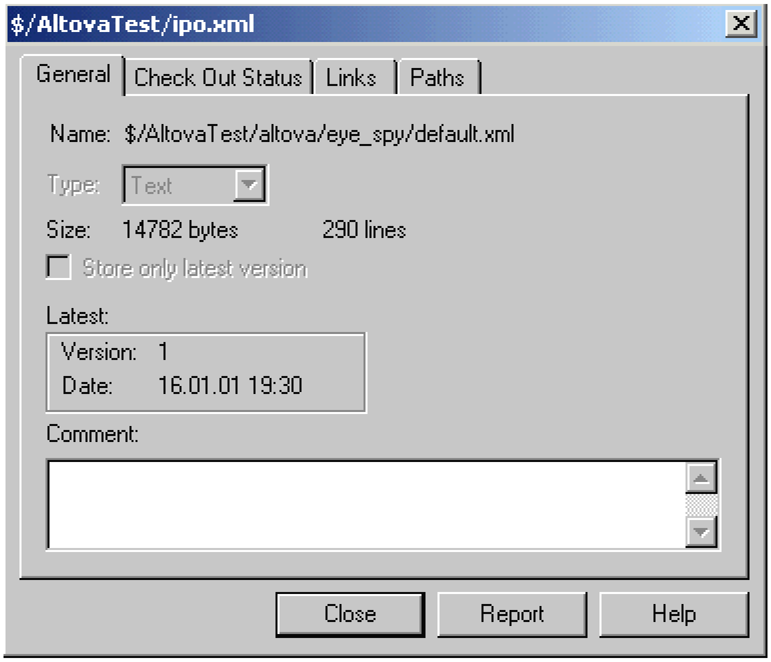
Properties

This command **displays** the properties of the currently selected file, and is dependent on the source control provider you use.

To display the properties of the currently selected file:

- Select **Project** | **Source Control** | **Properties**.

This command can only be used on single files.



$/AltovaTest/ipo.xml

Name: $/AltovaTest/altova/eye_spy/default.xml

Type: Text

Size: 14782 bytes 290 lines

□ Store only latest version

Latest:

Version: 1

Date: 16.01.01 19:30

Comment:

Refresh Status

This command **refreshes** the status of all project files independent of their current status.

Run Native Interface

This command **starts** your source control software with its usual user interface.

11.3.7 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.

11.3.8 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.

11.3.9 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.

11.3.10 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.

11.3.11 Add Project Folder to Project...

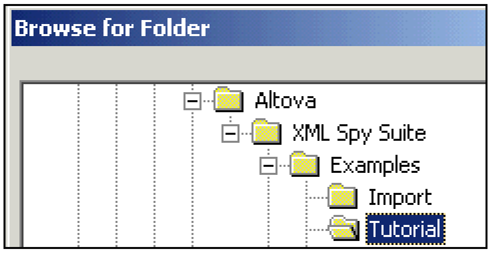
The **Project | Add Project Folder to Project...** command adds a new folder to the current project. Use this command to add a new folder to the current project or a sub-folder to a project folder. You can also access this command from the context-menu when you right-click on a folder in the project window.

11.3.12 Add External Folder to Project...

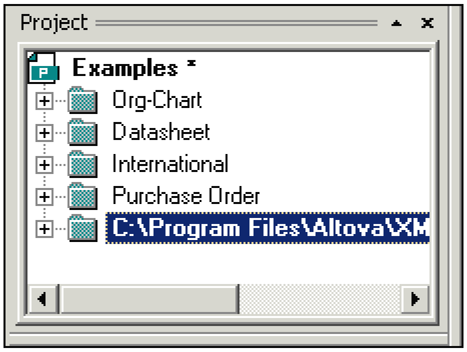
The **Project | Add External Folder to Project** command adds a new external folder to the current project. Use this command to add a local or network folder to the current project. You can also access this command from the context-menu when you right-click a folder in the project window.
Please note: Files contained in external folders cannot be placed under source control.

Adding external folders to projects
To add an external folder to the project:

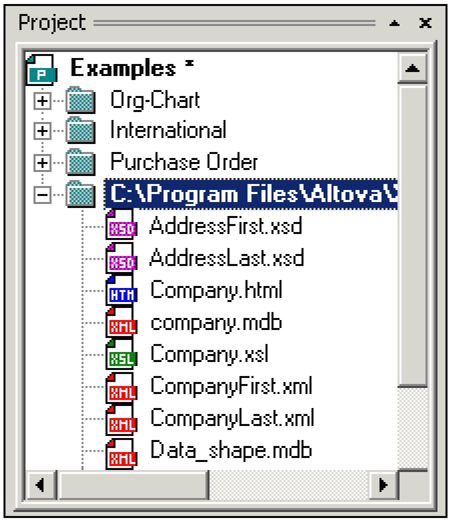
1. Select the menu option **Project | Add External Folder to Project**.
2. Select the folder you want to include from the Browse for Folder dialog box, and click **OK** to confirm.



The selected folder now appears in the project window.



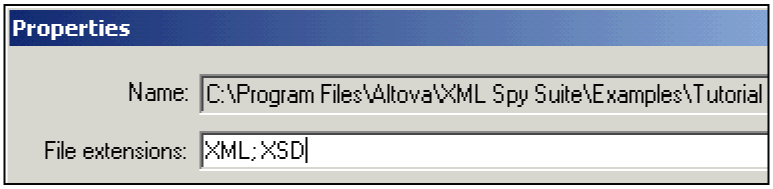
3. Click the plus icon to view the folder contents.



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.



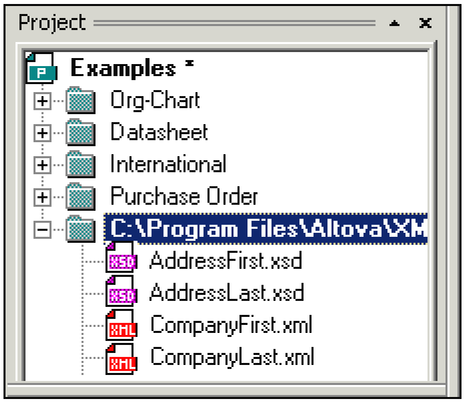
Name: C:\Program Files\Altova\XML Spy Suite\Examples\Tutorial

File extensions: XML; XSD

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.



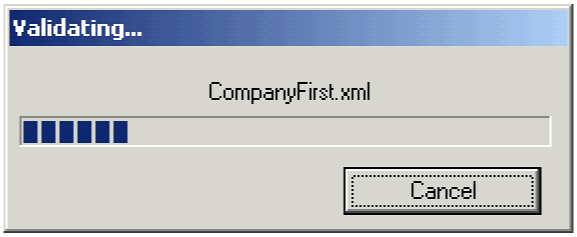
- Examples *
  - Org-Chart
  - Datasheet
  - International
  - Purchase Order
  - C:\Program Files\Altova\XM
    - AddressFirst.xsd
    - AddressLast.xsd
    - CompanyFirst.xml
    - CompanyLast.xml

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:

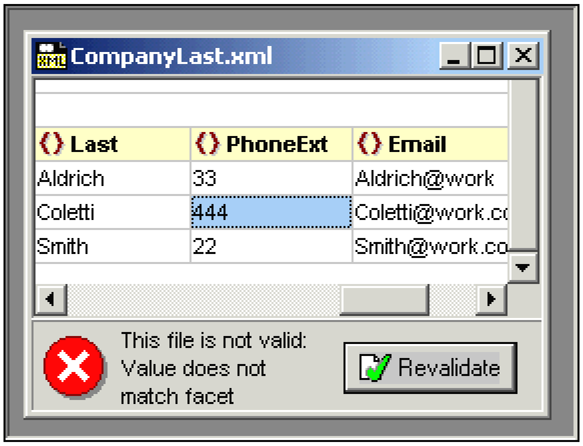
1. Select the file types you want to see or check from the external folder,
2. Click the folder and click the **Check well-formedness** or **Validate** icon (hotkeys **F7** or **F8**). All the files visible under the folder are checked.



Validating...

CompanyFirst.xml

If a file is malformed or invalid, then this file is opened in the Main Window, allowing you to edit it. In this case the CompanyLast file is opened because the PhoneExt value does not match the facet defined in the underlying schema file. This schema only allows two-digit numbers.



Last	PhoneExt	Email
Aldrich	33	Aldrich@work
Coletti	444	Coletti@work.co
Smith	22	Smith@work.co

This file is not valid: Value does not match facet

3. Correct the error and restart the process to recheck the rest of the folder.

Please note: You can select discontinuous files in the folder, by holding down CTRL and clicking the files singly. Only these files are then checked when you click F7 or F8.

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 files or folders:

- Right-click a **folder** and press the **Delete** key to delete the folder from the Project window. This only deletes the folder from the Project view, and does not delete anything on your hard disk or network.
- Right-clicking a **single file** and pressing **Delete does not delete** a file from the Project window. You have to delete it physically and then Refresh the external folder contents.

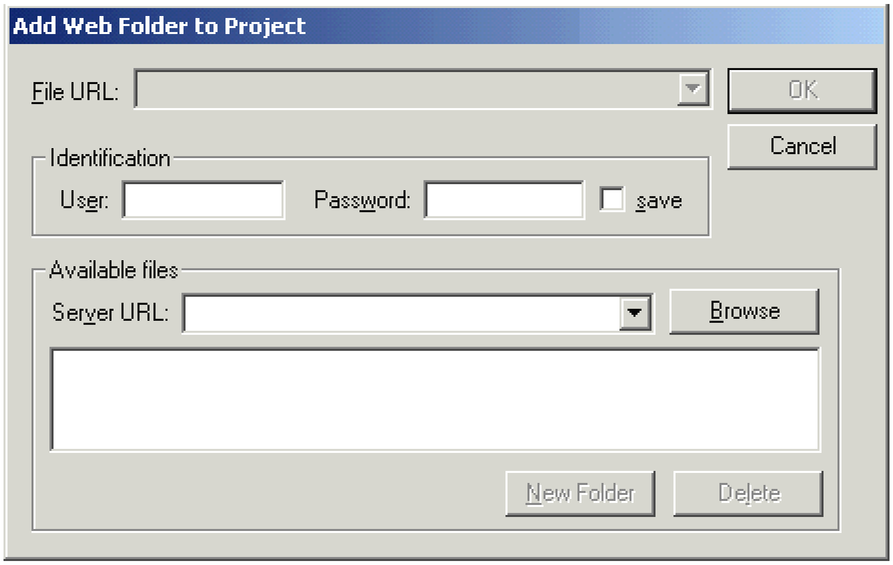
11.3.13 Add External Web Folder to Project...

This command **adds** a new **external web folder** to the current project.

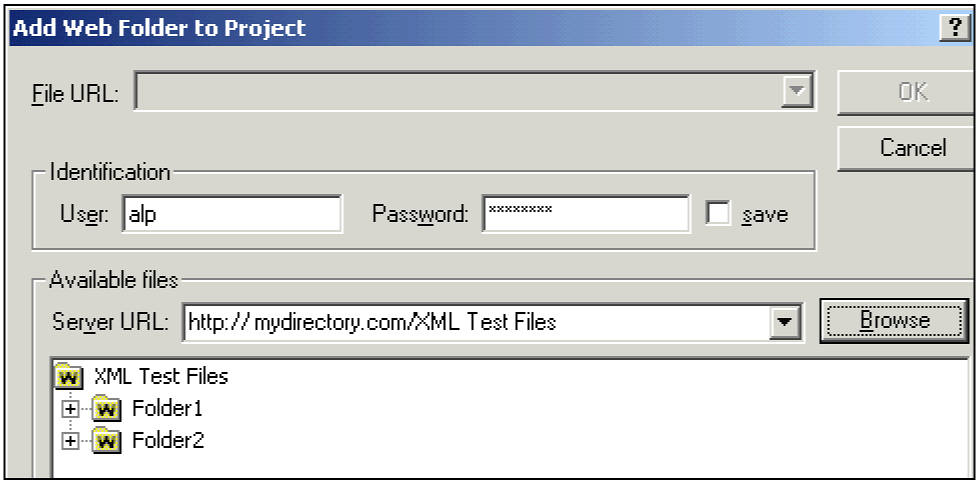
Use this command to add a web folder to the current project. You can also access this command from the context-menu when you right-click a folder in the project window. Please note that files contained in external folders cannot be placed under source control.

To add an external web folder to the project:

1. Select the menu option **Project | Add External Web Folder to project**.
   This opens the Add Web folder to project dialog box.



2. Click in the Server URL field to enter the server URL, and enter the login ID in the User and Password fields.
3. Click **Browse** to connect to the server and view the files available there.

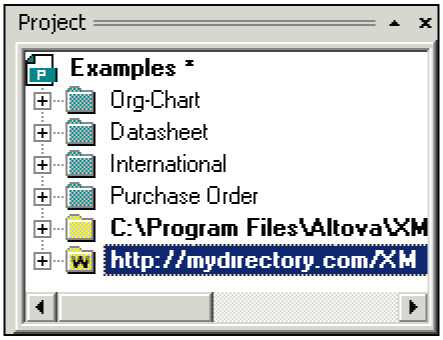


User: alp

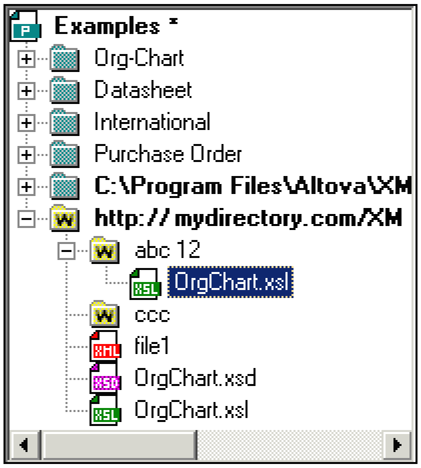
Password: ********

Server URL: http:// mydirectory.com/XML Test Files

4. Click the **folder** you want to add to the project view. The **OK** button only becomes active once you do this. The Folder name and http: server address now appear in the **File URL** field.
5. Click **OK** to add the folder to the project.



6. Click the plus icon to view the folder contents.



To filter the folder contents:

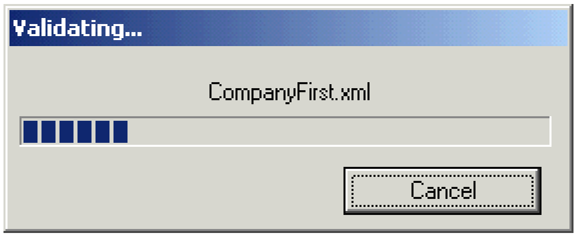
1. **Right click** the web folder, and select the popup menu option **Properties**. This opens the Properties dialog box.
2. Click in the **File extensions** field and enter the file extensions of the file types you want to see. You can separate each file type with a **semicolon** to define multiple types (XML

and Schema XSDs for example).

3. Click **OK** to confirm.
   The Project window now only shows the XML and XSD files of the web folder.

Validating and checking a folder for well-formedness:

1. 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.



Validating...

CompanyFirst.xml

   If a file is malformed or invalid, then this file is opened in the main window, allowing you to edit it.
2. Correct the error and restart the process to recheck the rest of the folder.

Please note:

You can select discontinuous files in the folder, by holding down CTRL and clicking the files singly. Only these files are then checked when you click **F7** or **F8**.

To update the project folder contents:

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 popup menu option **Refresh external folder**.

Deleting files or folders:

- Right-click a **folder** and hit the **Delete** key, to delete the **folder** from the Project window. This only deletes the folder from the Project view, and does not delete anything on the web server.
- Right-clicking a **single file** and hitting **Delete does not delete** a file from the Project window. You have to delete it physically and then Refresh the external folder contents.

11.3.14 Project Properties...

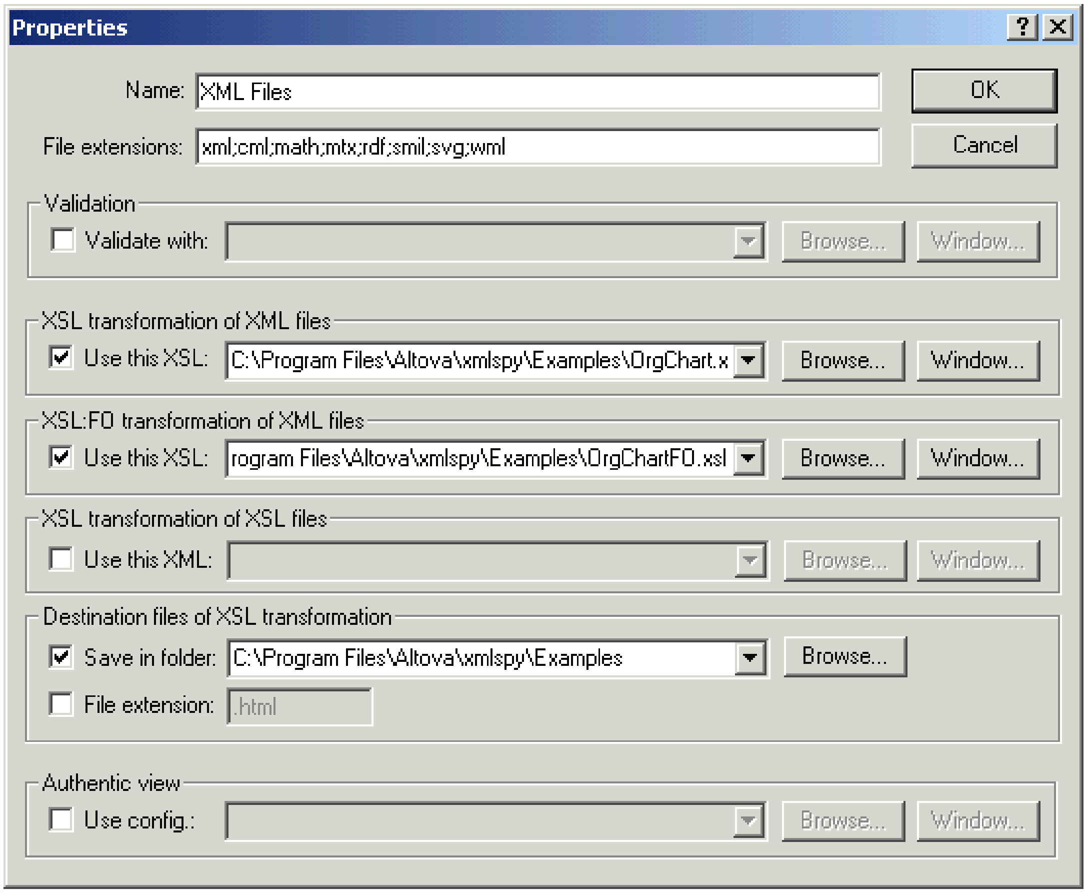
The **Project | Project Properties** command lets you define important settings for any of the specific folders in your project.

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:

XSL transformation of XML files

☑ Use this XSL: C:\Program Files\Altova\xmlspy\Examples\OrgChart.x

XSL:FO transformation of XML files

☑ Use this XSL: rogram Files\Altova\xmlspy\Examples\OrgChartFO.xsl

XSL transformation of XSL files

□ Use this XML:

Destination files of XSL transformation

☑ Save in folder: C:\Program Files\Altova\xmlspy\Examples

□ File extension: .html

Authentic view

□ Use config.:

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).

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.

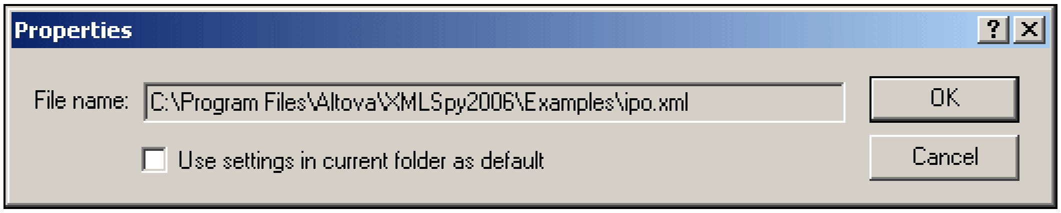
Destination files of XSL transformation

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.



File name: C:\Program Files\Altova\XMLSpy2006\Examples\ipo.xml

☐ Use settings in current folder as default

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.

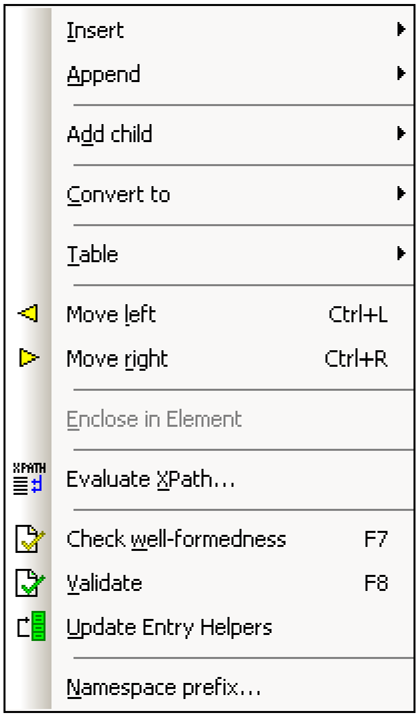
11.3.15 Most Recently Used Projects

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).

11.4 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 XPaths in the context of individual XML documents.

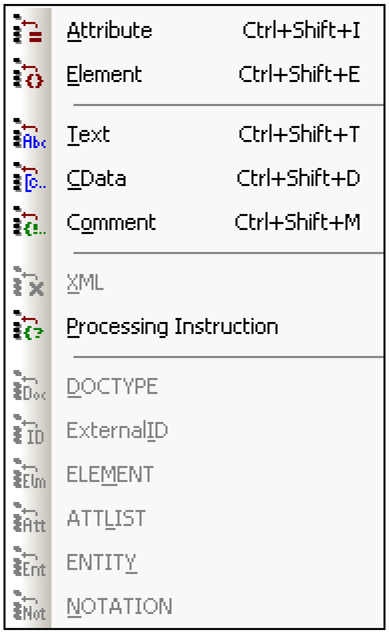


Among the most frequently used XML tasks are checks for the well-formedness of documents and validity of XML documents. Commands for these tasks are in this menu.

11.4.1 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.



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"?>
```

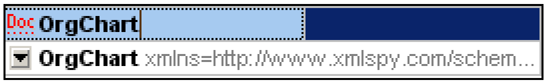
Please note: Since an XML document may only contain one XML declaration at the very top of the file, this command should only be used with the topmost row selected and if an XML declaration does not already exist.

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 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.



Doc OrgChart

OrgChart xmlns=http://www.xmlspy.com/schem...

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.

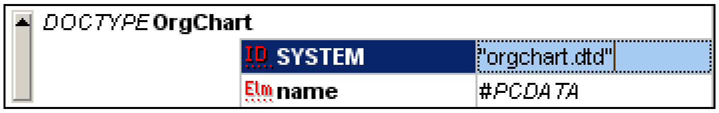
Insert ExternalID

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 a catalog file called `MainCatalog.xml`. A public identifier in an XML document can be used to dereference a DTD listed in `MainCatalog.xml`.

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.



DOCTYPE OrgChart

ID SYSTEM "orgchart.dtd"

Elm name *#PCDATA*

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.

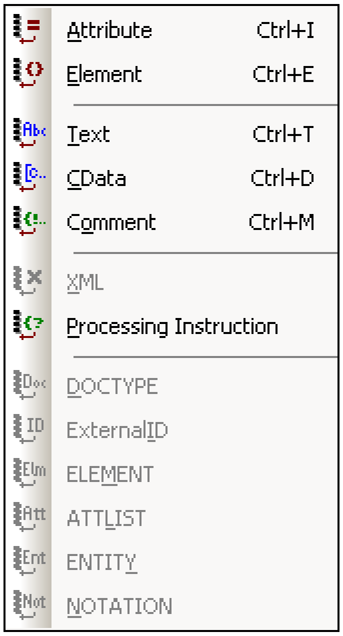
Insert NOTATION

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.

11.4.2 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.



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.

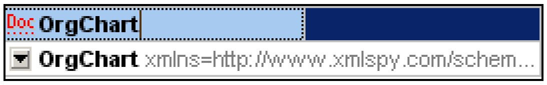
Please note: An XML document may contain only one XML declaration, which must appear at the very top of the file.

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 comment node is appended after the last child of the selected attribute's parent element.

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.



Doc OrgChart

OrgChart xmlns=http://www.xmlspy.com/schem...

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.

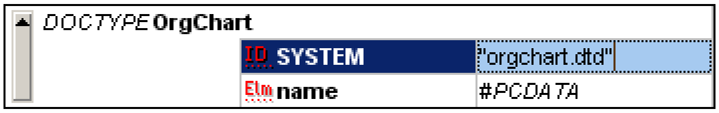
Append ExternalID

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 a catalog file called `MainCatalog.xml`. A public identifier in an XML document can be used to dereference a DTD listed in `MainCatalog.xml`.

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.



DOCTYPE OrgChart

ID SYSTEM	"orgchart.dtd"
Elm name	#PCDATA

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.

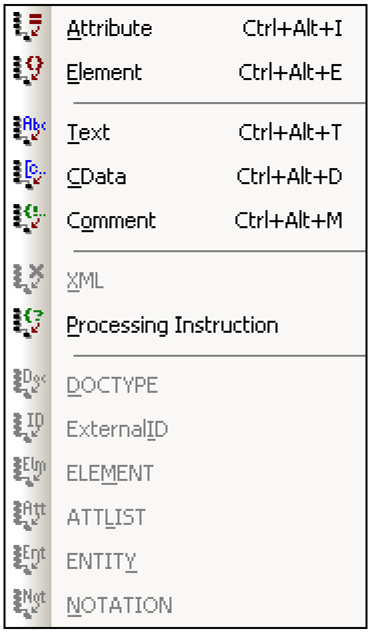
Append NOTATION

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.

11.4.3 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

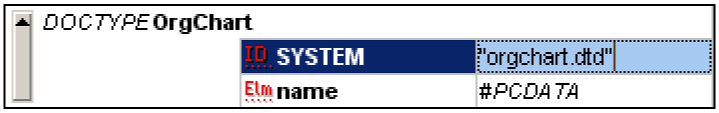
The **XML** | **Add Child | Processing Instruction** command is available in Grid View only and inserts a new Processing Instruction (PI) as a child of the selected item.

Add Child 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.



DOCTYPE **OrgChart**

ID **SYSTEM**	"orgchart.dtd"
Elm **name**	*#PCDATA*

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

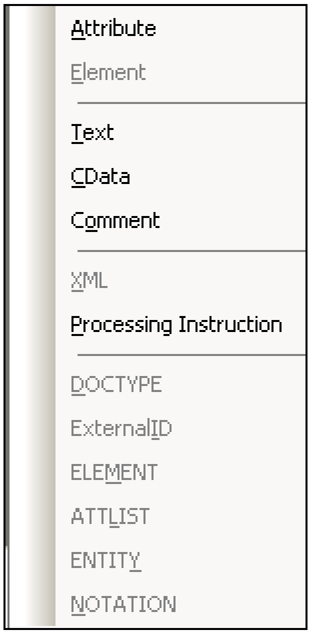
The **XML | Add Child | ENTITY** command is available in Grid View only, for DTD documents, or when the DOCTYPE declaration of an XML document is selected. It appends an ENTITY declaration to the list of declarations.

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.

11.4.4 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

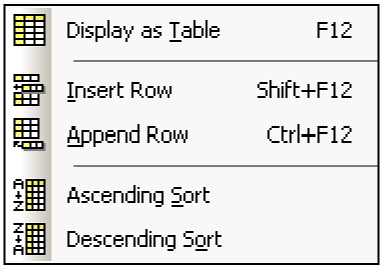
The **XML | Convert to | ENTITY** command converts the selected item to an entity declaration in a DOCTYPE declaration or in an external DTD.

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.

11.4.5 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.



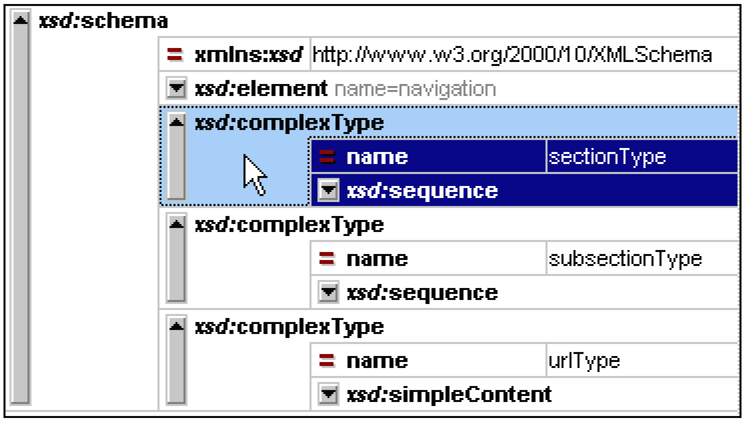
Display as Table

▦ **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).

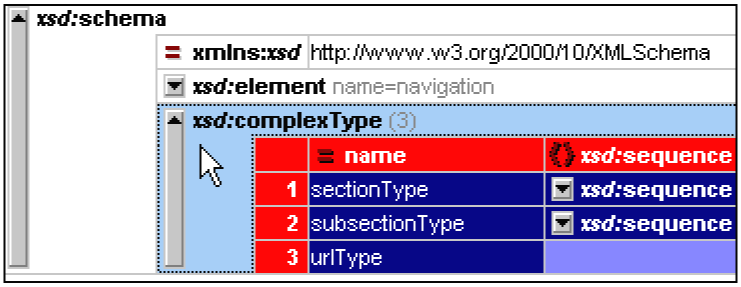
To switch to Table View:

1. Select any one occurrence of the repeating element you wish to view as a table.



```
xsd:schema
    = xmlns:xsd  http://www.w3.org/2000/10/XMLSchema
    xsd:element name=navigation
    xsd:complexType
        = name  sectionType
        xsd:sequence
    xsd:complexType
        = name  subsectionType
        xsd:sequence
    xsd:complexType
        = name  urlType
        xsd:simpleContent
```

2. Click **XML | Display as Table** or **F12** or the ▦ toolbar icon.



xsd:schema

= xmlns:xsd http://www.w3.org/2000/10/XMLSchema

xsd:element name=navigation

xsd:complexType (3)

	= name	xsd:sequence
1	sectionType	xsd:sequence
2	subsectionType	xsd:sequence
3	urlType	

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.

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

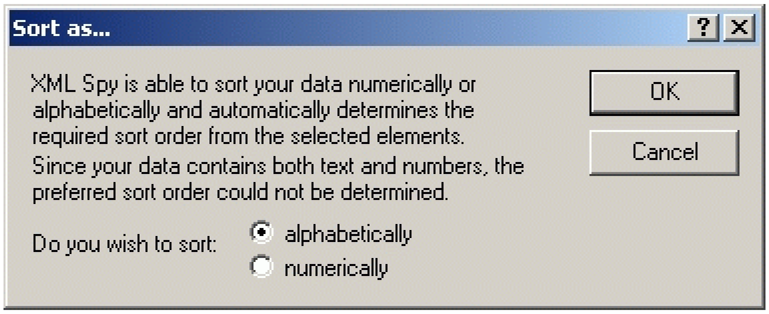
Append Row

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*).



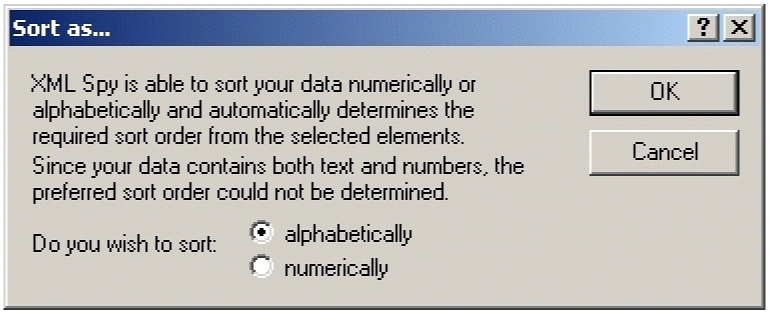
Sort as...

XML Spy is able to sort your data numerically or alphabetically and automatically determines the required sort order from the selected elements.
Since your data contains both text and numbers, the preferred sort order could not be determined.

Do you wish to sort: ● alphabetically
○ numerically

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*).



XML Spy is able to sort your data numerically or alphabetically and automatically determines the required sort order from the selected elements.

Since your data contains both text and numbers, the preferred sort order could not be determined.

Do you wish to sort: ● alphabetically ○ numerically

11.4.6 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.

11.4.7 Move Right

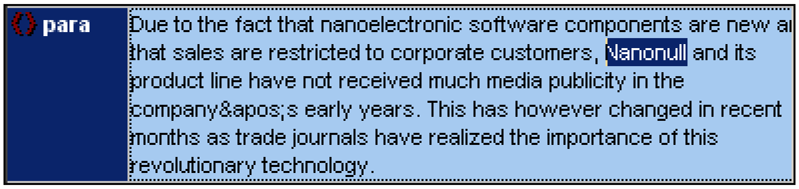
▷ **Ctrl+R**

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.

11.4.8 Enclose in Element

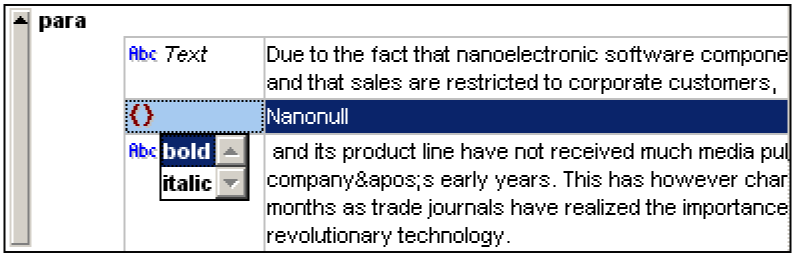
The **XML** | **Enclose in Element** command is enabled in Grid View only. It encloses a selected text range in a new element. The new element is created inline around the selected text. If you are editing a document based on a Schema or DTD, you will automatically be presented with a list of valid choices 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.



Due to the fact that nanoelectronic software components are new ai that sales are restricted to corporate customers, Nanonull and its product line have not received much media publicity in the company&apos;s early years. This has however changed in recent months as trade journals have realized the importance of this revolutionary technology.

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`.



para

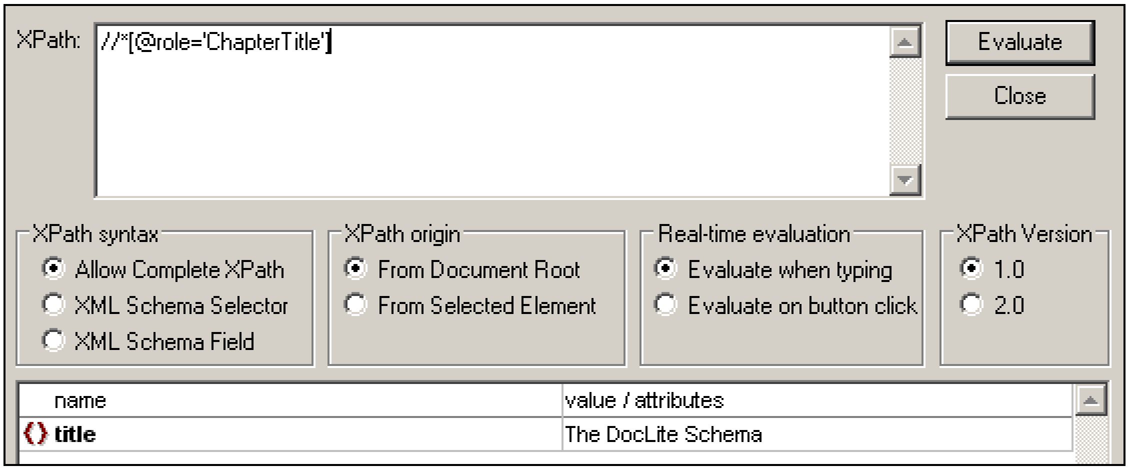
Abc *Text*	Due to the fact that nanoelectronic software compone and that sales are restricted to corporate customers,
()	Nanonull
Abc bold italic	and its product line have not received much media pul company&apos;s early years. This has however char months as trade journals have realized the importance revolutionary technology.

The selection you make will be the name of the new element. Alternatively, you can enter some other name for the element.

11.4.9 Evaluate XPath

The **XML | Evaluate XPath...** command pops up the Evaluate XPath dialog (*shown below*). This dialog enables you to evaluate XPath 1.0 or XPath 2.0 expressions for the active XML document. The **Evaluate XPath...** command is enabled in Text View and Grid View.

The Evaluate XPath dialog consists of three areas: an input pane, an options area, and a results pane. You enter an XPath expression in the input pane, and the evaluation result is displayed in the results pane. The result is a list of items in the result sequence, together with their corresponding values. Selecting an item in the list causes that item to be highlighted in the Main Window.



XPath: //*[@role='ChapterTitle']

XPath syntax
- ● Allow Complete XPath
- ○ XML Schema Selector
- ○ XML Schema Field

XPath origin
- ● From Document Root
- ○ From Selected Element

Real-time evaluation
- ● Evaluate when typing
- ○ Evaluate on button click

XPath Version
- ● 1.0
- ○ 2.0

name	value / attributes
() **title**	The DocLite Schema

XPath expressions in the input pane

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 input pane, 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 XPath Evaluator options. So set the options correctly, especially the XPath Version and the XPath Origin.
- Only namespaces that are in scope on the element where the XPath originates (i.e. on the context node) are evaluated. You can set the origin (or context node) of the XPath expression to be the Document Root or the element in which the cursor is placed. Only

those namespaces that are in scope on the context node (or the Document Element when the Document Root is selected) should be used in the XPath expression to be evaluated.

- The XPath Evaluator uses two different engines for the two versions of the language, so there are no backward compatibility issues.

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 **should not** specify a prefix on functions. If you do, 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 file); `fn:current-date()` (with Functions namespace declared in XML file and bound to prefix `fn:`). You can omit the namespace prefix even if the Functions namespace has been declared in the XML file; this is because the function is then in the default namespace (which is the Functions namespace).
- 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 namespaces 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 are not 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.

XPath Evaluator options

The following evaluation options are available.

- XPath syntax: For XPath 1.0 evaluations, you can select whether to use the full XPath 1.0 grammar, the XML Schema Selector grammar, or the XML Schema Field grammar.
- XPath origin: Should the context node be the document node (or document root) or a selected node.
- Real-time evaluation: Should evaluation results be displayed as you type or when you click the **Validate** button.
- XPath Version: Select whether XPath 1.0 or XPath 2.0 should be used to evaluate the expression.

Please note: Options are retained for subsequent evaluations till changed.

Troubleshooting

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

1. Check whether the options have been correctly set.
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

Given below is a list of built-in XPath 2.0 functions that have limited support.

Function	Support limitation
`fn:lower-case`	The ASCII character set is supported.
`fn:normalize-unicode`	Not supported.

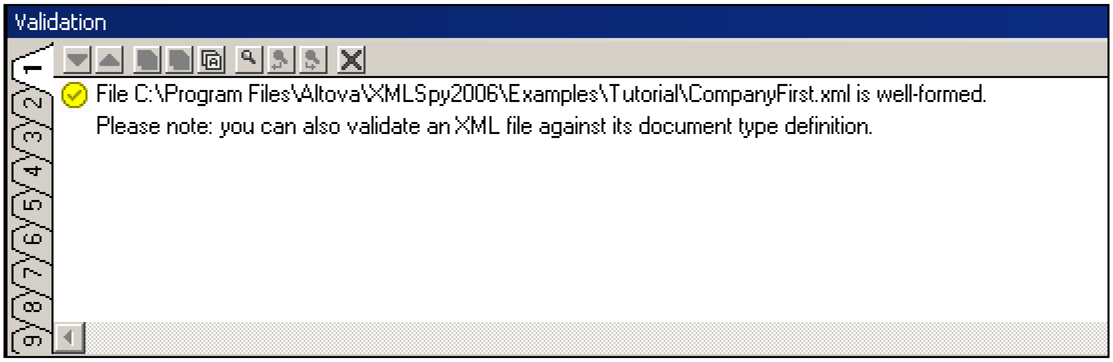
`fn:upper-case`	The ASCII character set is supported.

11.4.10 Check Well-Formedness

F7

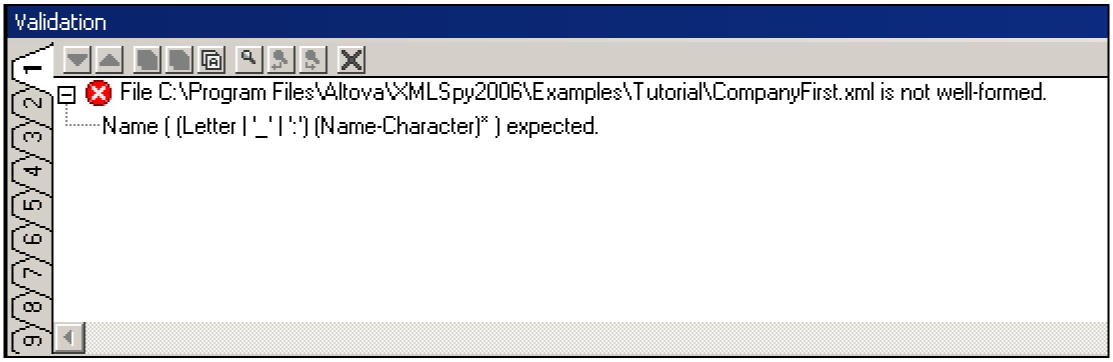
The **XML | Check well-formedness (F7)** command checks the active document for well-formedness by the definitions of the XML 1.0 specification. Every XML document **must** be well-formed. XMLSpy checks for well-formedness whenever a document is opened or saved, or when the view is changed from Text 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:



File C:\Program Files\Altova\XMLSpy2006\Examples\Tutorial\CompanyFirst.xml is well-formed.
Please note: you can also validate an XML file against its document type definition.

If an error is encountered during the well-formedness check, a corresponding error message is displayed:



File C:\Program Files\Altova\XMLSpy2006\Examples\Tutorial\CompanyFirst.xml is not well-formed.
Name ((Letter | '_' | ':') (Name-Character)*) expected.

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).

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.

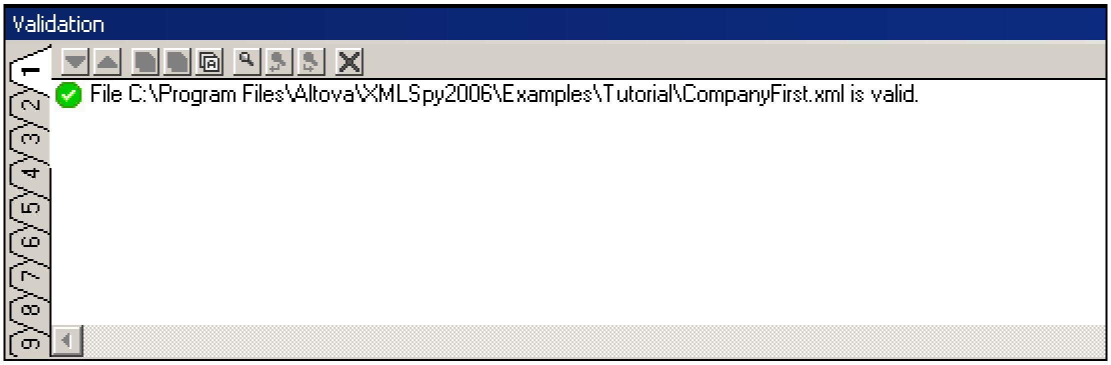
Please note: You can also use the **Check well-formedness** command on any file, folder, or group of files in the active project window. Click on the respective folder, and then on the Check Well-Formedness icon.

11.4.11 Validate

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.

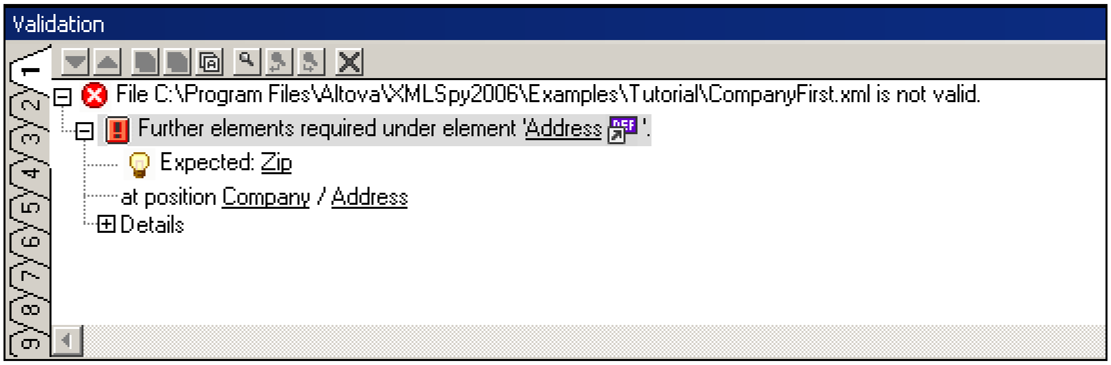
If a document is valid, a successful validation message is displayed in the Validation window:



✓ File C:\Program Files\Altova\XMLSpy2006\Examples\Tutorial\CompanyFirst.xml is valid.

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.



File C:\Program Files\Altova\XMLSpy2006\Examples\Tutorial\CompanyFirst.xml is not valid.
Further elements required under element 'Address'.
Expected: Zip
at position Company / Address
Details

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 at the bottom of the Main 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**.

XMLSpy supports a subset of the OASIS XML catalogs mechanism. These catalogs enable 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 be changed in the catalog files only.) The mechanism works as follows:

- A catalog file matches a `PUBLIC` system identifier to a URI that points to a local file. In XMLSpy, the catalog lookup is performed by two files: `MainCatalog.xml` and `CustomCatalog.xml`. `MainCatalog.xml` maps pre-defined `PUBLIC` system identifiers of several popular schemas to URIs that point to the locally saved schema. `CustomCatalog.xml` enables you to make your own catalog extensions. Both `MainCatalog.xml` and `CustomCatalog.xml` are installed with your XMLSpy application package. The schemas, stylesheets, and other files referenced from `MainCatalog.xml` are located in specific subfolders of your XMLSpy installation folder.
- The `PUBLIC` 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 lookup to be changed when XML documents are used on multiple machines. In XMLSpy, `MainCatalog.xml` is looked up first and then `CustomCatalog.xml`.

When writing your `CustomCatalog.xml` file, use only the following subset of the OASIS catalog in order for XMLSpy to process the catalog correctly.

```
<catalog...>
<public publicId="nnn" uri="mmm"/>
<uri name="nnn" uri="mmm"/>
<system systemId="nnn" uri="mmm"/>
<rewriteURI uriIdStartString="nnn" rewritePrefix="mmm"/>
<rewriteSystem systemIdStartString="nnn" rewritePrefix="mmm"/>
```

Please note:

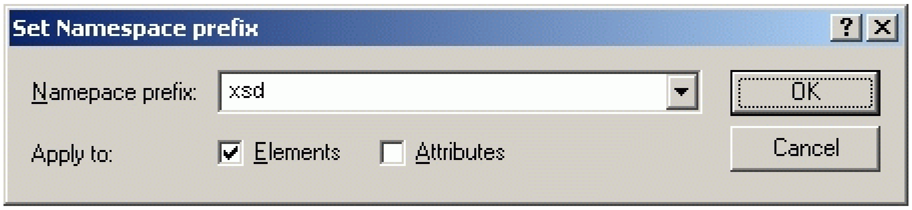
- Although the DTDs for XML Schemas are referenced from `MainCatalog.xml`, you cannot validate your XML files against either of these schemas. The purpose of these two DTDs is to provide entry helper info for editing purposes, should you wish to create files according to these older recommendations.
- For more information on catalogs, see the `XML Catalogs specification`.

11.4.12 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.

11.4.13 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.



Set Namespace prefix

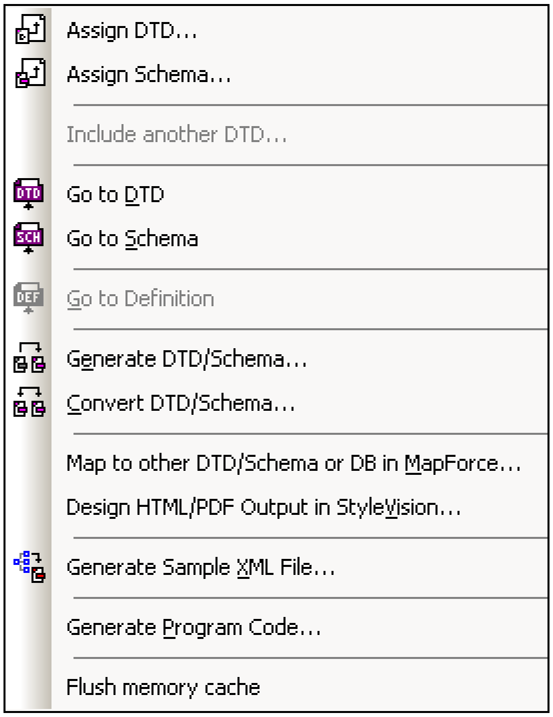
Namepace prefix: xsd

Apply to: ☑ Elements ☐ Attributes

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.

11.5 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.

11.5.1 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. 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.

11.5.2 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. 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"
```

11.5.3 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.

11.5.4 Go to DTD

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.

11.5.5 Go to Schema

SCH

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.

11.5.6 Go to Definition

DEF

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 Enhanced Grid View

1. Click left on the item.
2. Select the menu item **DTD/Schema | Go to Definition,** or click on the icon.

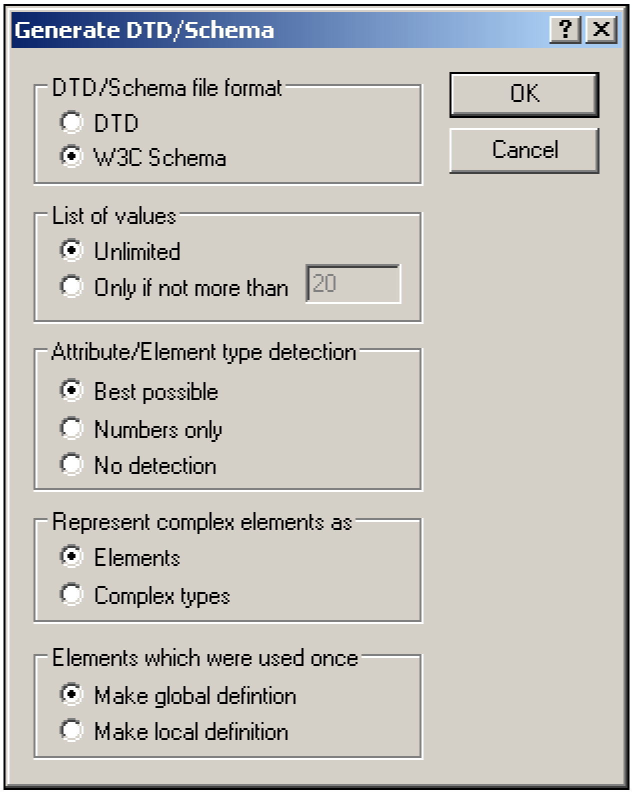
To see the item definition in Schema/WSDL Design View

1. Use CTRL + Double click on the item you want to see the definition of, or
2. 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.

11.5.7 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 an XML document.



Generate DTD/Schema

DTD/Schema file format
○ DTD
● W3C Schema

List of values
● Unlimited
○ Only if not more than 20

Attribute/Element type detection
● Best possible
○ Numbers only
○ No detection

Represent complex elements as
● Elements
○ Complex types

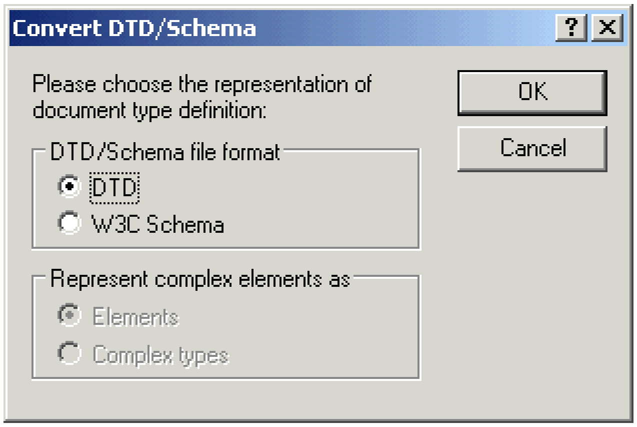
Elements which were used once
● Make global defintion
○ Make local definition

If you generate an XML Schema, XMLSpy automatically detects the datatypes used in the XML document/s and creates the corresponding types in the schema. XMLSpy will also optionally detect typical enumeration scenarios (List of Values), where an element or attribute can only contain items from a predefined list of values. You can also decide how to represent complex elements and how elements that only appear only once should be treated.

This 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.

11.5.8 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.



Convert DTD/Schema

Please choose the representation of document type definition:

DTD/Schema file format

● DTD

○ W3C Schema

Represent complex elements as

● Elements

○ 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.

11.5.9 Map to other DTD/Schema or DB in MapForce...

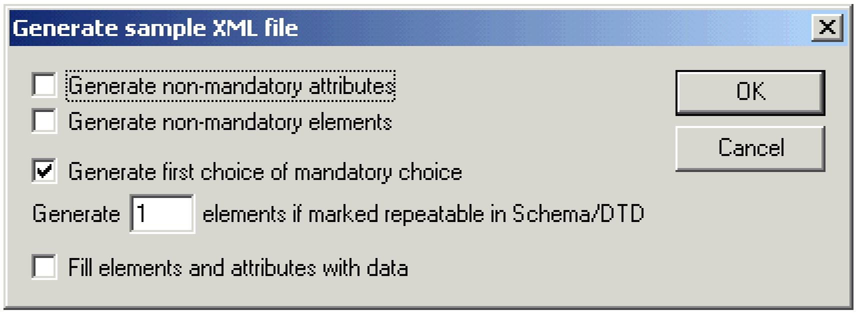
The **DTD/Schema | Map to other DTD/Schema or DB in MapForce** command launches Altova's MapForce if the application is installed. MapForce enables you to map a schema to another DTD, XML Schema, or database.

11.5.10 Design HTML/PDF Output in 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.

11.5.11 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 non-mandatory attributes
□ Generate non-mandatory elements
☑ Generate first choice of mandatory choice
Generate 1 elements if marked repeatable in Schema/DTD
□ Fill elements and attributes with data

Generate non-mandatory attributes
Activating this option generates not only mandatory attributes, but also the non-mandatory attributes, defined in the schema.

Generate non-mandatory elements
Activating this option generates not only mandatory, but also non-mandatory elements, defined in the schema.

Generate first choice of mandatory choice
Activating this option generates/inserts the first choice of a mandatory choice.

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.

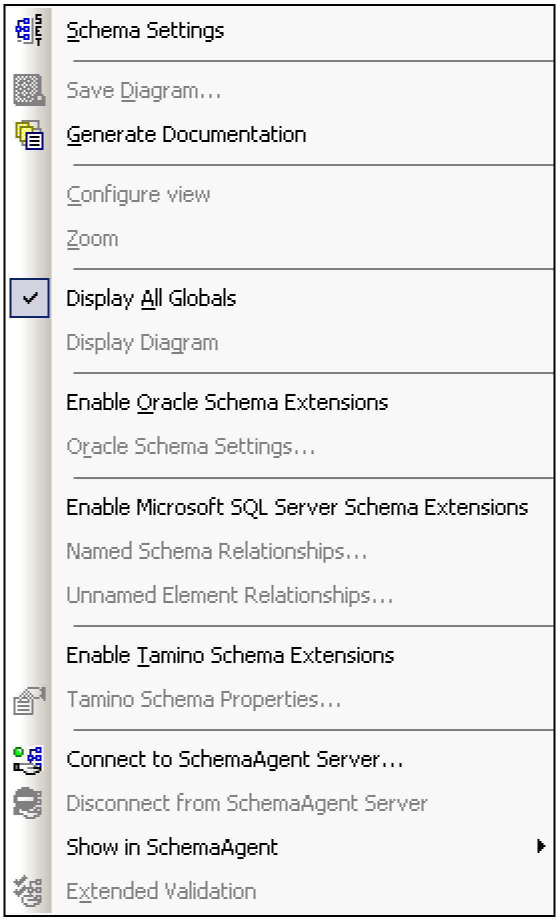
11.5.12 Flush Memory Cache

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.

11.6 Schema Design Menu

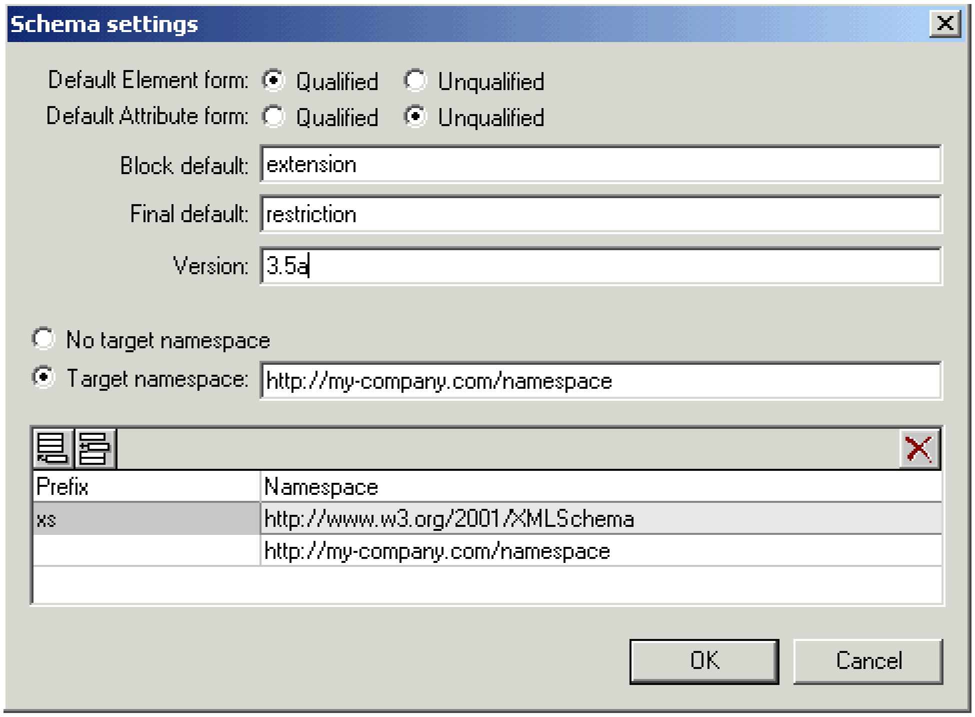
The **Schema Design** menu enables you to configure the Schema Design View of XMLSpy. This view enables you to design XML Schemas in a GUI. It is available when an XML Schema document is active in Schema/WSDL View.


Schema Settings
Save Diagram...
Generate Documentation
Configure view
Zoom
Display All Globals
Display Diagram
Enable Oracle Schema Extensions
Oracle Schema Settings...
Enable Microsoft SQL Server Schema Extensions
Named Schema Relationships...
Unnamed Element Relationships...
Enable Tamino Schema Extensions
Tamino Schema Properties...
Connect to SchemaAgent Server...
Disconnect from SchemaAgent Server
Show in SchemaAgent
Extended Validation

The commands available in this menu are described in this section.

11.6.1 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: extension

Final default: restriction

Version: 3.5a

○ No target namespace

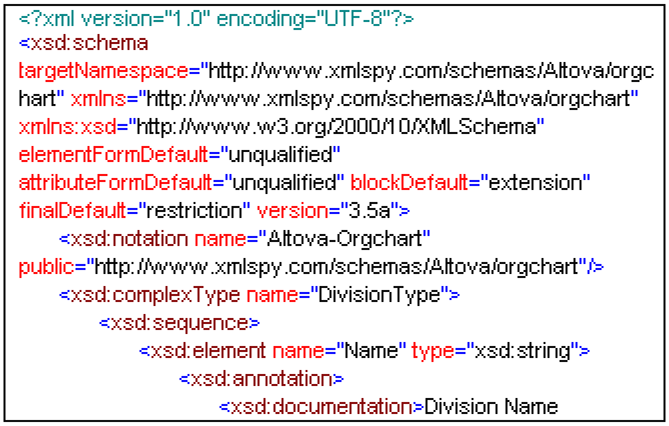
● Target namespace: http://my-company.com/namespace

Prefix	Namespace
xs	http://www.w3.org/2001/XMLSchema
	http://my-company.com/namespace

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:

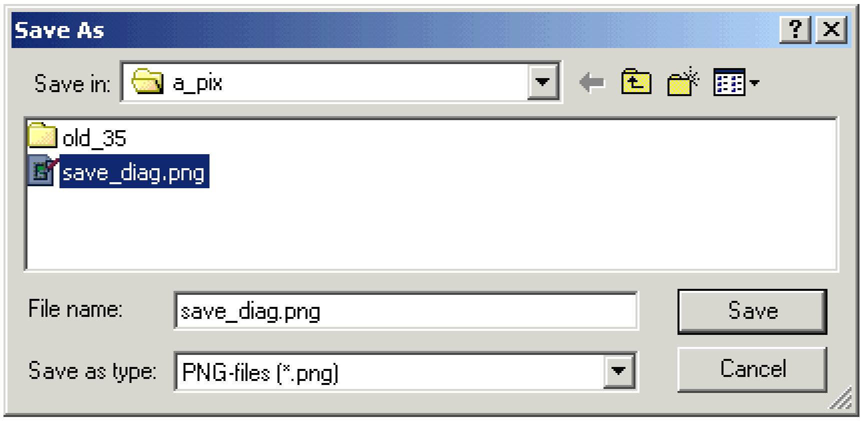


```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
targetNamespace="http://www.xmlspy.com/schemas/Altova/orgc
hart" xmlns="http://www.xmlspy.com/schemas/Altova/orgchart"
xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
elementFormDefault="unqualified"
attributeFormDefault="unqualified" blockDefault="extension"
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.

11.6.2 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.


Save As

Save in: a_pix

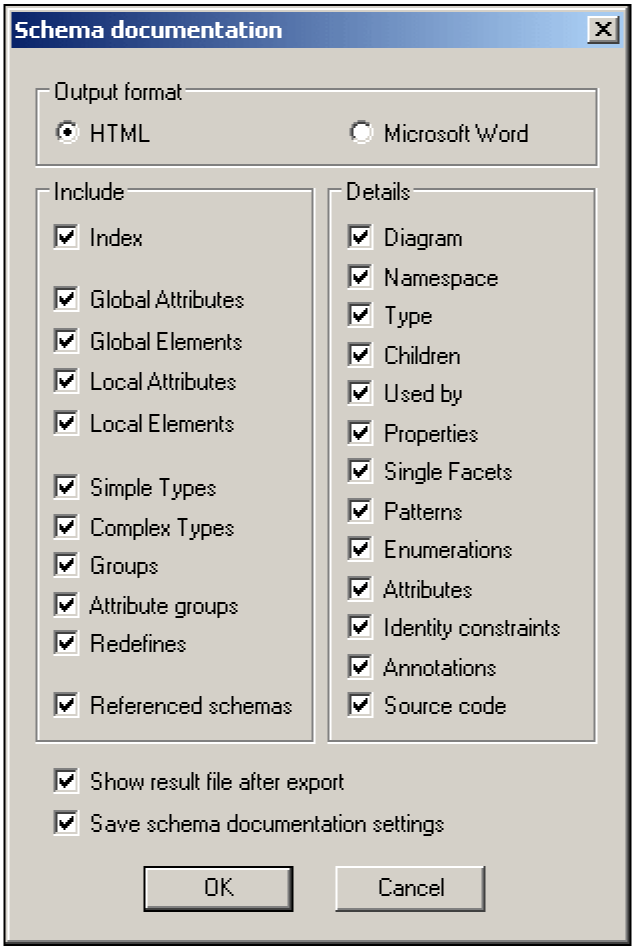
old_35
save_diag.png

File name: save_diag.png

Save as type: PNG-files (*.png)

11.6.3 Generate Documentation

The **Schema Design | Generate Documentation** command generates detailed documentation about your schema in HTML or MS Word. Documentation is generated for components you select in the Schema Documentation dialog (which appears when you select the Generate Documentation command; *see screenshot*). Related elements (child elements, complex types, etc.) are 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 also that schema documentation is also generated for **included schema components**.



Schema documentation

Output format

● HTML ○ Microsoft Word

Include

☑ Index

☑ Global Attributes

☑ Global Elements

☑ Local Attributes

☑ Local Elements

☑ Simple Types

☑ Complex Types

☑ Groups

☑ Attribute groups

☑ Redefines

☑ Referenced schemas

Details

☑ Diagram

☑ Namespace

☑ Type

☑ Children

☑ Used by

☑ Properties

☑ Single Facets

☑ Patterns

☑ Enumerations

☑ Attributes

☑ Identity constraints

☑ Annotations

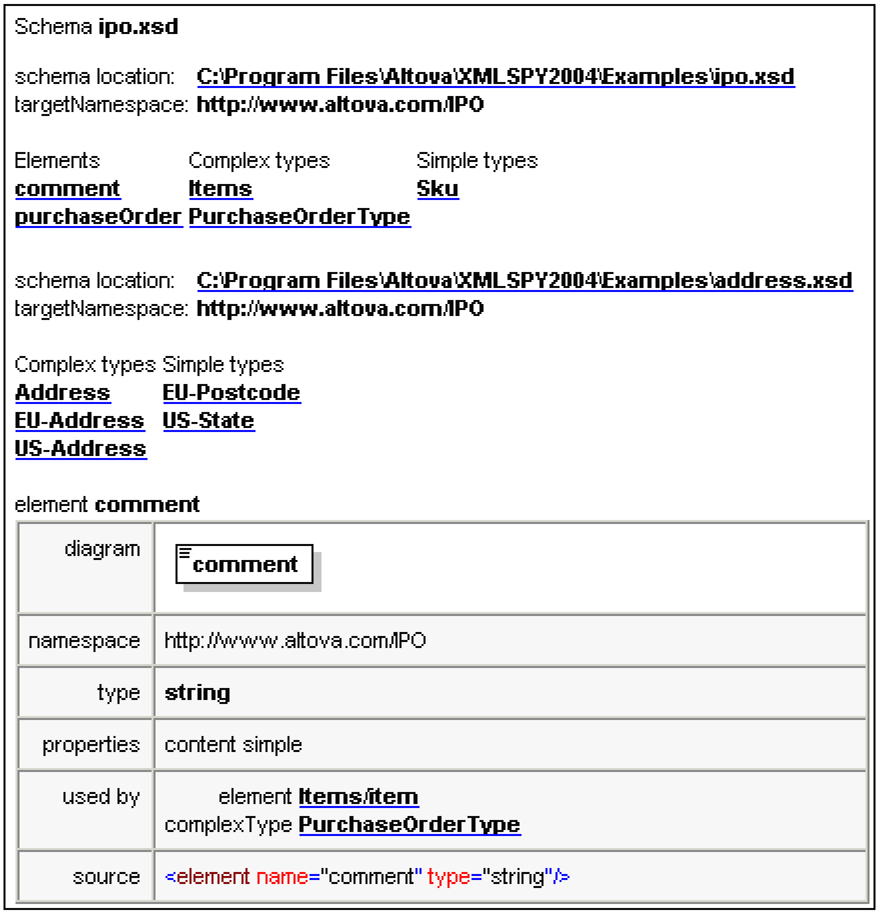
☑ Source code

☑ Show result file after export

☑ Save schema documentation settings

Clicking the **Generate Documentation** command opens the Schema documentation dialog:

- 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 **Details** pane lists the details that may be included for each component. Select the details you wish to include in the documentation. The Diagram option includes the graphical representation of each schema item in the document.
- If you opt for showing the result file after export, a result HTML file will display in the Browser View while a result Word file will open in MS Word.



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	

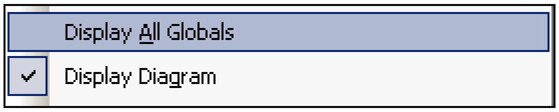
element **comment**

diagram	**comment**
namespace	http://www.altova.com/IPO
type	**string**
properties	content simple
used by	element **Items/item** complexType **PurchaseOrderType**
source	<element name="comment" type="string"/>

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.

11.6.4 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.

11.6.5 Configure view...

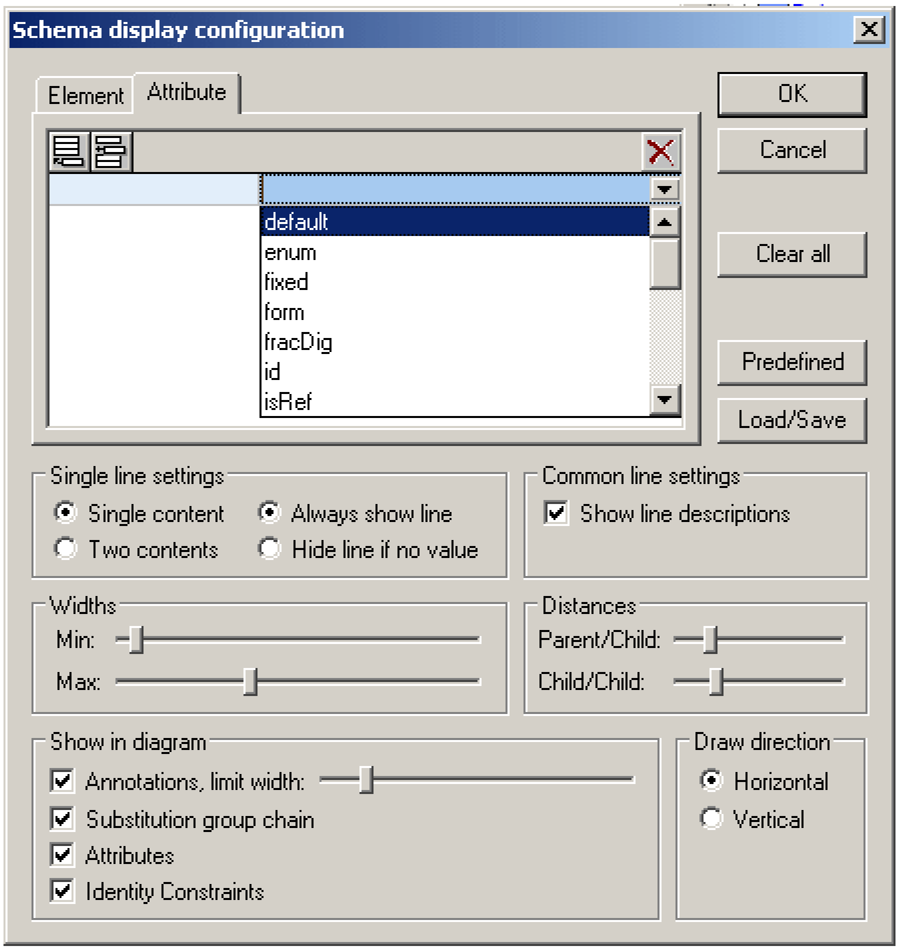
The **Schema Design | Configure view** command allows you to configure the Content Model View. Clicking the command opens the Schema Display Configuration dialog at the bottom right of the XMLSpy window, enabling you to see the effect of your settings as you enter them in the dialog. The settings take 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 1 and 2 for as many properties as required.



Schema display configuration

Element | Attribute

default
enum
fixed
form
fracDig
id
isRef

Single line settings
● Single content ○ Two contents
● Always show line ○ Hide line if no value

Common line settings
☑ Show line descriptions

Widths
Min:
Max:

Distances
Parent/Child:
Child/Child:

Show in diagram
☑ Annotations, limit width:
☑ Substitution group chain
☑ Attributes
☑ Identity Constraints

Draw direction
● Horizontal
○ Vertical

The Content Model View is updated, showing the defined property descriptor lines for all elements for which they exist.

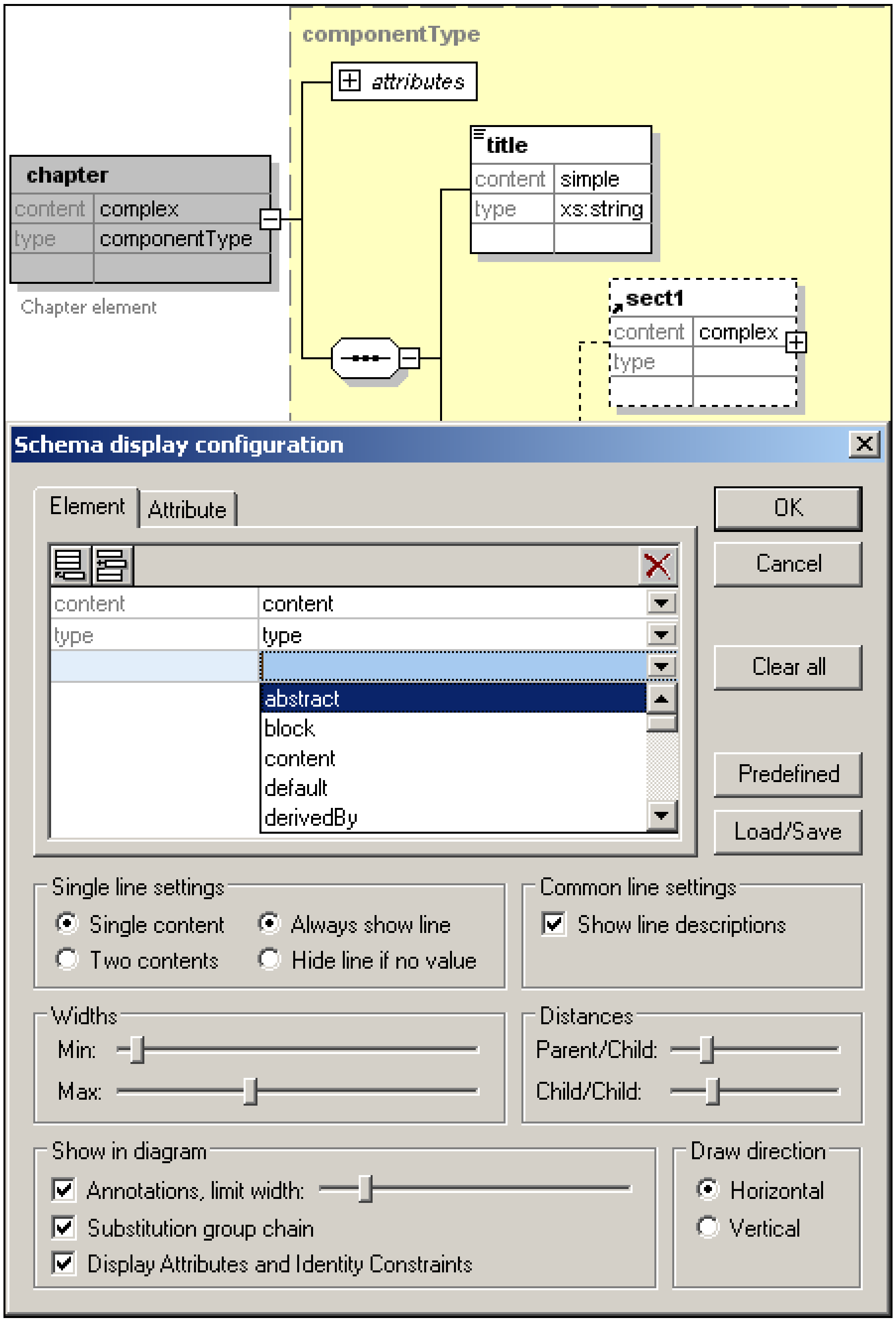
Please note:

- 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.



content	content
type	type

abstract
block
content
default
derivedBy

Single line settings
● Single content ● Always show line
○ Two contents ○ Hide line if no value

Common line settings
☑ Show line descriptions

Widths
Min:
Max:

Distances
Parent/Child:
Child/Child:

Show in diagram
☑ Annotations, limit width:
☑ Substitution group chain
☑ Display Attributes and Identity Constraints

Draw direction
● Horizontal
○ Vertical

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 this check box is 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.

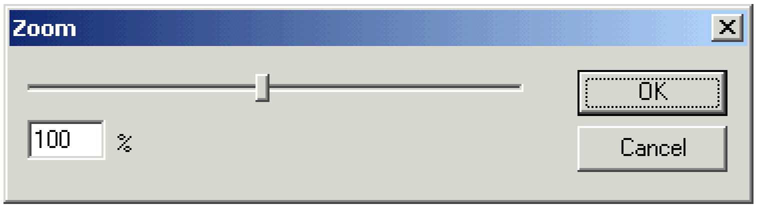
Editing the content model in the diagram itself
You can change element properties directly in the content model diagram. To do this, double-click the property you wish to edit and start entering data. If a selection is available, a drop-down list appears, from which you can select an option. Otherwise, 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 retrieve 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.

11.6.6 Zoom

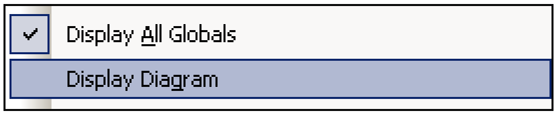
The **Schema Design | Zoom** command controls the zoom factor of the Content Model View. This feature is useful if you have a large content model and wish to zoom out so that the entire content model fits in the Main Window. You can zoom between 10% and 200% of actual size.



To zoom in and out, either drag the slider or click in the entry box and enter a percentage value.

11.6.7 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 (*screenshot below*).



Alternatively, you could use the following methods to switch to Content Model View:

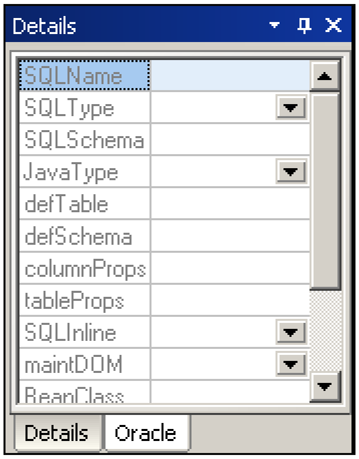
- Click the icon next to the component, the content model of which you want to display.
- Double-click a component name in the Component Navigator Entry Helper (at top right).

11.6.8 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`.

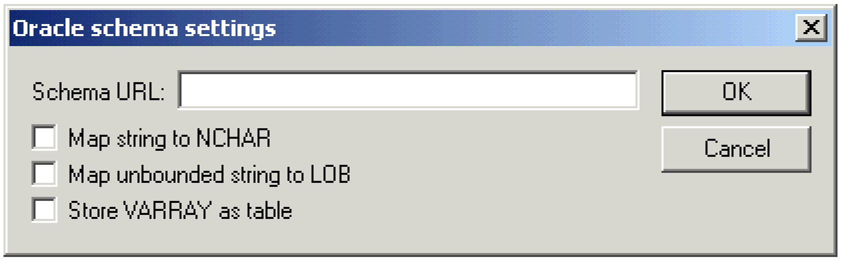


Oracle extensions can be defined for complex types, elements, and attributes. Use the Entry Helper as you normally would in XMLSpy.

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.

11.6.9 Oracle Schema Settings

The **Schema Design | Oracle Schema Settings** command allows you to define global settings for Oracle schema extensions.



Oracle schema settings

Schema URL:

☐ Map string to NCHAR

☐ Map unbounded string to LOB

☐ Store VARRAY as table

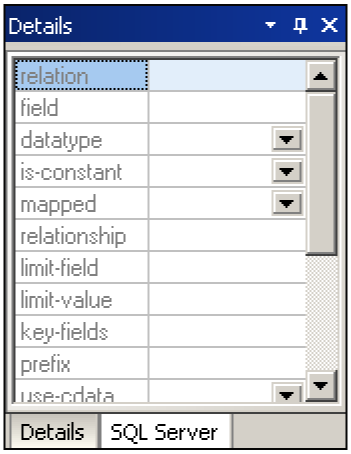
In order to access this dialog, Oracle schema extensions must be enabled (using the Enable Oracle Schema Extensions command).

11.6.10 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.

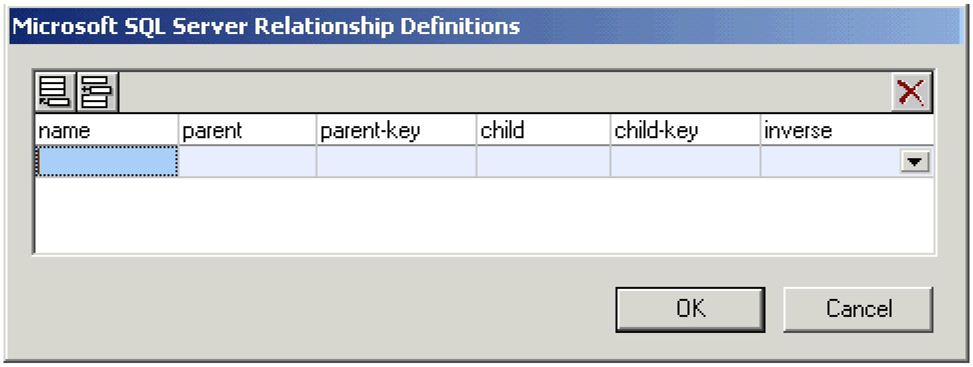
Please note: This menu command can be toggled on and off, that is, extensions can be enabled or disabled. When SQL Server extensions are enabled, the command is displayed with a check mark to its left. Disabling SQL Server extensions (by clicking the enabled command) deletes the SQL Server namespace declaration and all SQL extensions in the file. A warning message appears since this action cannot be undone.

11.6.11 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.



Microsoft SQL Server Relationship Definitions

name	parent	parent-key	child	child-key	inverse

This generates a SQL relationship element, placing it just after the namespace declaration.

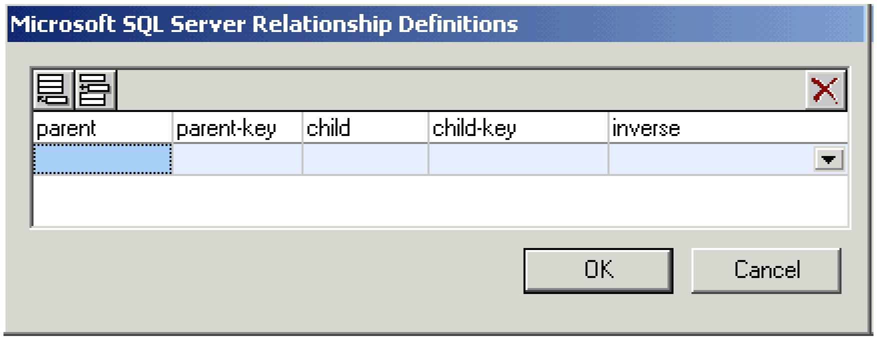
Please note: Click the delete icon , to delete a row from the dialog box.

11.6.12 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.

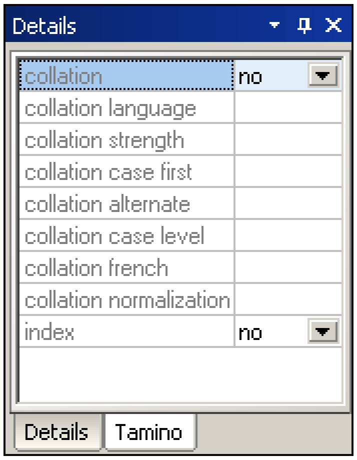
Please note: Click the delete icon ☒, to delete a row from the dialog box.

11.6.13 Enable Tamino Schema Extensions

XMLSpy provides support for Tamino schema extensions for use with Tamino Server. Software AG and Altova GmbH have entered into a worldwide agreement giving XMLSpy users easy and low-cost access to Tamino Server through a Tamino database bundle. For more information, see the Tamino section of this documentation. Using the Tamino schema extensions allows you to configure and customize how Tamino Server stores XML documents. These XML documents are then accessible through queries and legacy tools.

When you select the **Schema Design | Enable Tamino Schema Extensions** command, the following occurs:

- If the XML Schema namespace prefix is anything other than `xs:`, XMLSpy asks whether it may change the prefix to `xs:`, as required by Tamino.
- If the file extension is anything other than `.tsd`, XMLSpy asks whether it may change the file extension to `.tsd`., as required by Tamino.
- The Tamino Properties dialog appears, enabling you to specify Tamino settings. See Tamino Properties for details.
- The Tamino namespace is declared on the `schema` element: `xmlns:tsd="http://namespaces.softwareag.com/tamino/TaminoSchemaDefinition"`.
- A Tamino tab is created in the Details Entry Helper, enabling you to add attributes to schema elements such as `xsd:element`.



collation	no
collation language	
collation strength	
collation case first	
collation alternate	
collation case level	
collation french	
collation normalization	
index	no

Where Tamino extensions can be defined for a schema component, the Tamino tab is available in the Details Entry Helper when the component is selected. Use the Entry Helper as you normally would in XMLSpy.

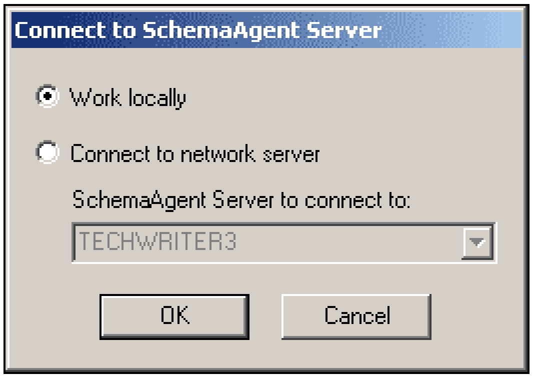
Please note: This menu command can be toggled on and off, that is, extensions can be enabled or disabled. When Tamino extensions are enabled, the command is displayed with a check mark to its left. Disabling Tamino extensions (by clicking the enabled command) deletes the Tamino namespace declaration and all Tamino extensions in the file. A warning message appears since this action cannot be undone.

11.6.14 Tamino Schema Properties

The **Schema Design | Tamino Schema Properties** command opens the Tamino Properties dialog box, in which you enter Tamino Database and Schema properties. See Tamino Schema Properties in the Tamino section of this documentation for details about these properties.

11.6.15 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:



Connect to SchemaAgent Server

● Work locally

○ Connect to network server

SchemaAgent Server to connect to: TECHWRITER3

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 Working with SchemaAgent in the Schema/WSDL View 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.

11.6.16 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.

11.6.17 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 Viewing schemas in SchemaAgent in the Schema/WSDL View section of this user manual.

11.6.18 Extended Validation

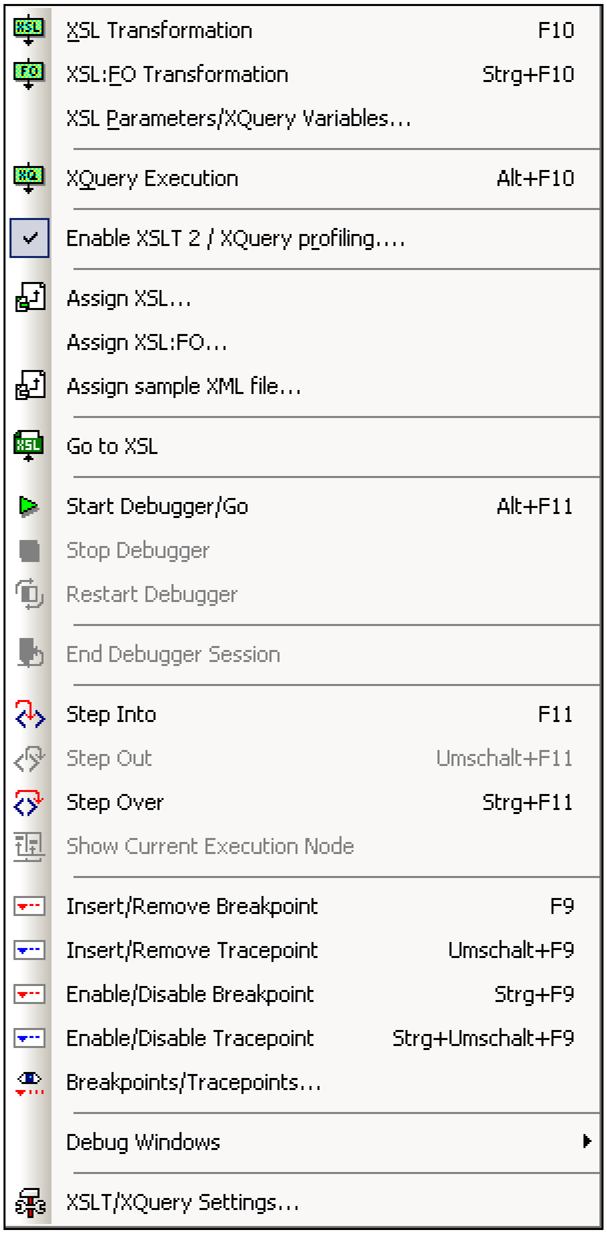
The **Extended Validation** command enables you to validate the currently active schema as well as schemas related to the currently active schema. This feature is described in detail in the Extended Validation section in the Schema/WSDL View of this user manual.

11.7 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.


XSL Transformation F10
XSL:FO Transformation Strg+F10
XSL Parameters/XQuery Variables...
XQuery Execution Alt+F10
Enable XSLT 2 / XQuery profiling....
Assign XSL...
Assign XSL:FO...
Assign sample XML file...
Go to XSL
Start Debugger/Go Alt+F11
Stop Debugger
Restart Debugger
End Debugger Session
Step Into F11
Step Out Umschalt+F11
Step Over Strg+F11
Show Current Execution Node
Insert/Remove Breakpoint F9
Insert/Remove Tracepoint Umschalt+F9
Enable/Disable Breakpoint Strg+F9
Enable/Disable Tracepoint Strg+Umschalt+F9
Breakpoints/Tracepoints...
Debug Windows
XSLT/XQuery Settings...

11.7.1 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. Alternatively, 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 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.

11.7.2 XSL:FO Transformation

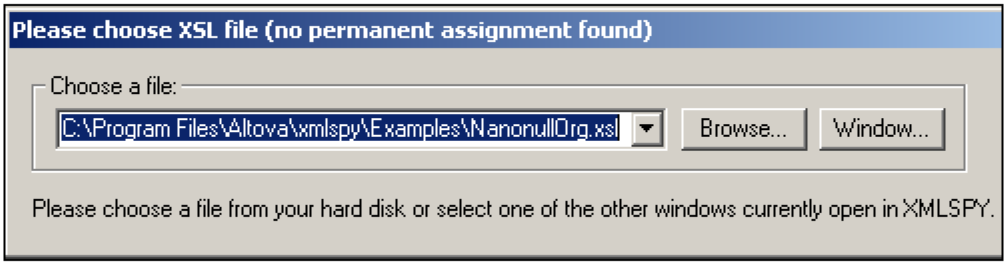
Ctrl+F10

FO is an XML format that describes paged documents. An FO processor, such as the Apache XML Project's FOP, consumes FO to generate PDF 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 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*).



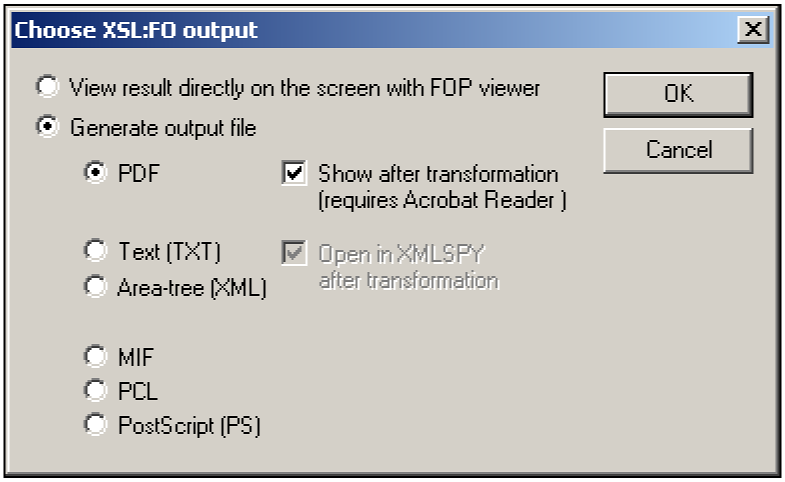
Please choose XSL file (no permanent assignment found)

Choose a file: C:\Program Files\Altova\xmlspy\Examples\NanonullOrg.xsl

Please choose a file from your hard disk or select one of the other windows currently open in XMLSPY.

  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.)



Choose XSL:FO output

○ View result directly on the screen with FOP viewer

● Generate output file

● PDF

☑ Show after transformation (requires Acrobat Reader)

○ Text (TXT)

○ Area-tree (XML)

☑ Open in XMLSPY after transformation

○ MIF

○ PCL

○ PostScript (PS)

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.

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:Transformation**. The XSLT stylesheet assigned to the selected project folder is used.

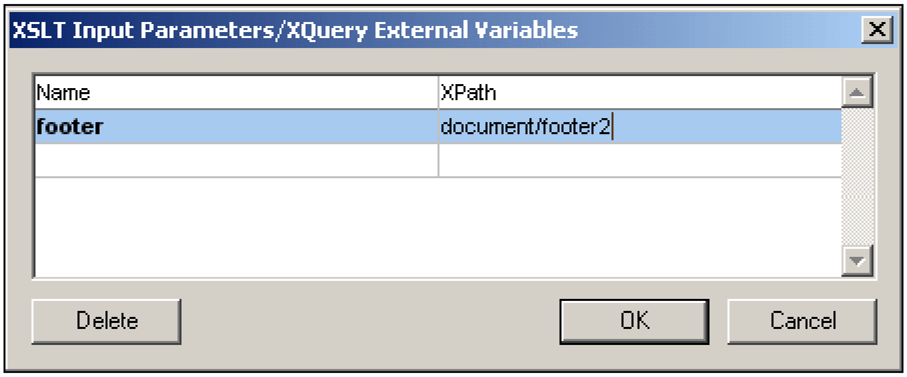
11.7.3 XSL Parameters/XQuery Variables...

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. The parameter value/s are passed to the parameter/s in the selected XSLT when you process an XML file using the **XSL Transformation** command in the XSL menu. External XQuery variable values are passed to the XQuery document when you select the **XQuery Execution** command.

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 without quotes or a text string delimited by quotes.



Name	XPath
footer	document/footer2

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 the XMLSpy has been closed.

Using 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:template match="/">
      <xsl:param name="footer" select="document/footer1" />
      <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>
```

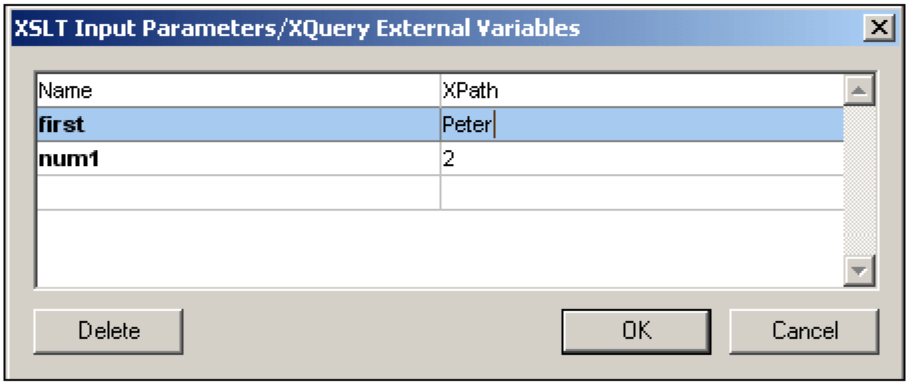
In the XSLT Input Parameters dialog, a new value for the `footer` parameter can be entered, such as the XPath: `document/footer2` (*see screenshot above*) or a text string. During transformation, this value is passed to the `footer` parameter in the template for the root element and is the value used when the footer block is instantiated.

Please 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 XSLT Input Parameters command will not be passed to the external processor.

Using external XQuery variables

The value you enter for an external XQuery variable must be a literal value without quotes. The datatype of the external variable is specified in the variable declaration in the XQuery document.



Name	XPath
first	Peter
num1	2

Please 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 transformations 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:

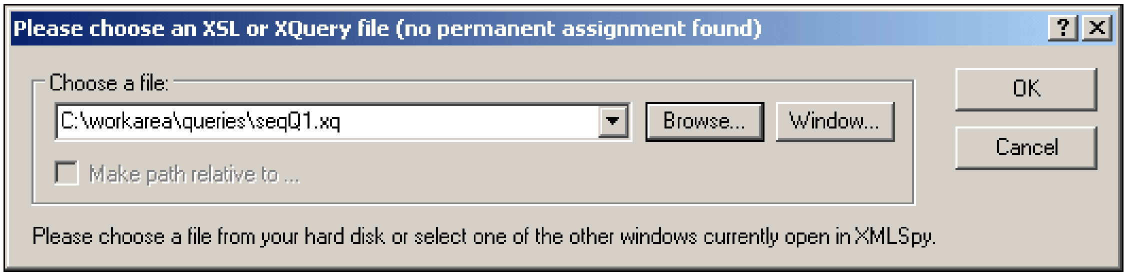
- The `external` keyword in the variable declaration in the XQuery document indicates that this variable is an external variable.
- Defining the static type of the variable is optional. If a datatype for the variable is not specified in the variable declaration, then the variable value is assigned the 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 )
  ```

Please 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.

11.7.4 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.



Please choose an XSL or XQuery file (no permanent assignment found)

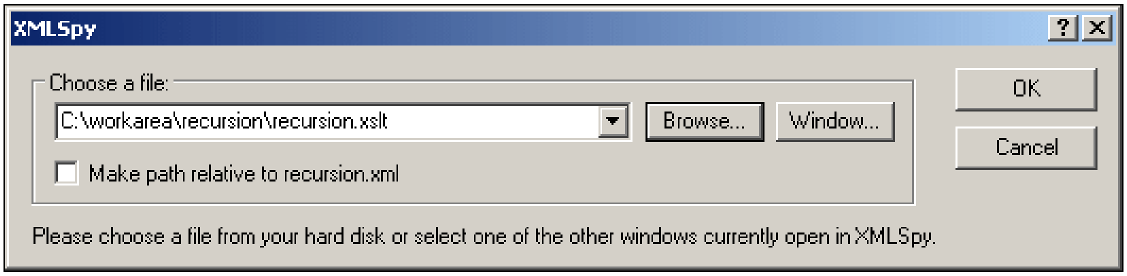
Choose a file: C:\workarea\queries\seqQ1.xq

□ Make path relative to ...

Please choose a file from your hard disk or select one of the other windows currently open in XMLSpy.

11.7.5 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.



Choose a file: C:\workarea\recursion\recursion.xslt

□ Make path relative to recursion.xml

Please choose a file from your hard disk or select one of the other windows currently open in XMLSpy.

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.

11.7.6 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.

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.

11.7.7 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.

11.7.8 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.

11.7.9 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.

11.7.10 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.

11.7.11 Restart Debugger

The **XSL/XQuery | Restart Debugger** command clears the output window and restarts the debugging session with the currently selected files.

11.7.12 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.

11.7.13 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.

11.7.14 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.

11.7.15 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.

11.7.16 Show Current Execution Nodes

The **XSL/XQuery | Show Current Execution Nodes** 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.

11.7.17 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.

11.7.18 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.

11.7.19 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.

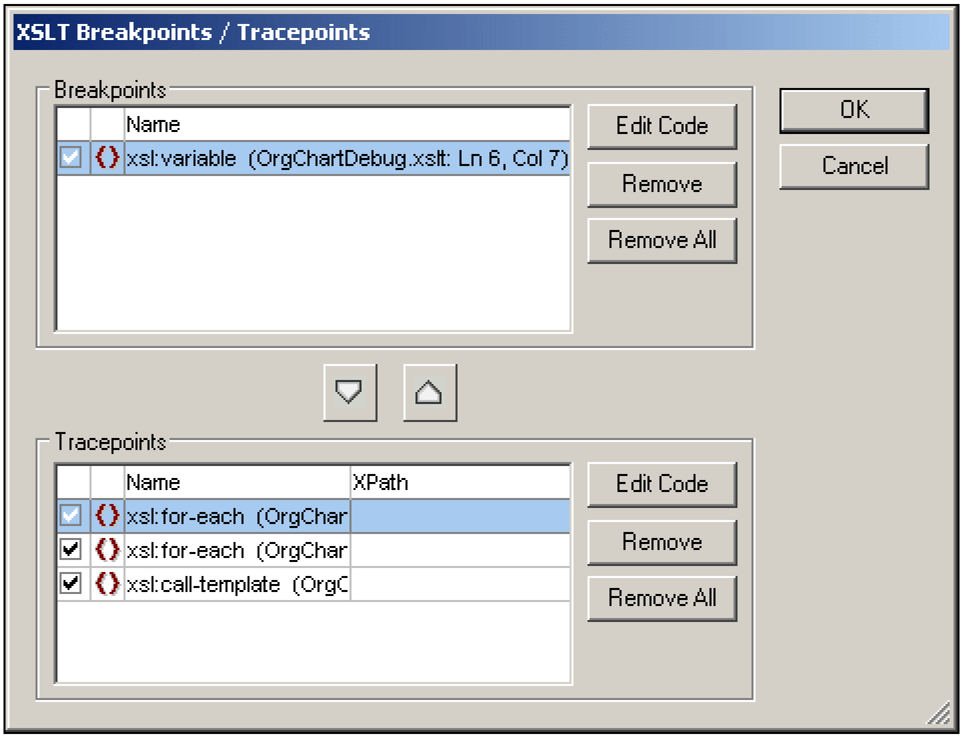
11.7.20 Enable/Disable Tracepoint

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.

11.7.21 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.



Breakpoints

		Name
☑	{}	xsl:variable (OrgChartDebug.xslt: Ln 6, Col 7)

Tracepoints

		Name	XPath
☑	{}	xsl:for-each (OrgChar	
☑	{}	xsl:for-each (OrgChar	
☑	{}	xsl:call-template (OrgC	

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.

11.7.22 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 highlights that Information Window in the XSLT/XQuery Debugger interface. This command can be used to effect only when a debugging session is in progress.

11.7.23 XSLT/XQuery Settings

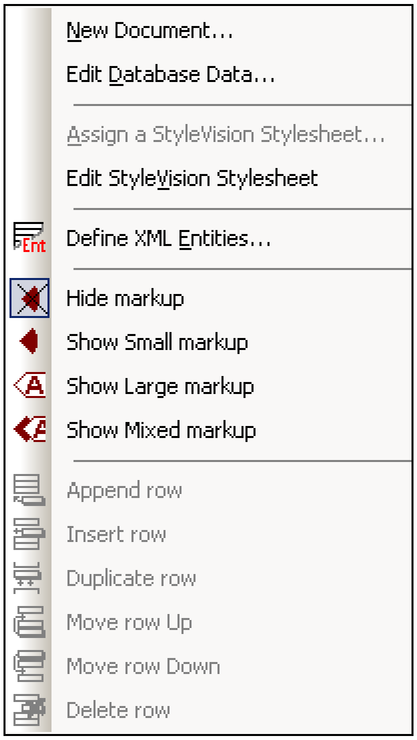
The **XSL/XQuery | XSLT/XQuery Settings** command opens the XSLT/XQuery Settings dialog, which enables you to set user options for the Debugger. See the XSLT/XQuery Debugger section for details.

11.8 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.

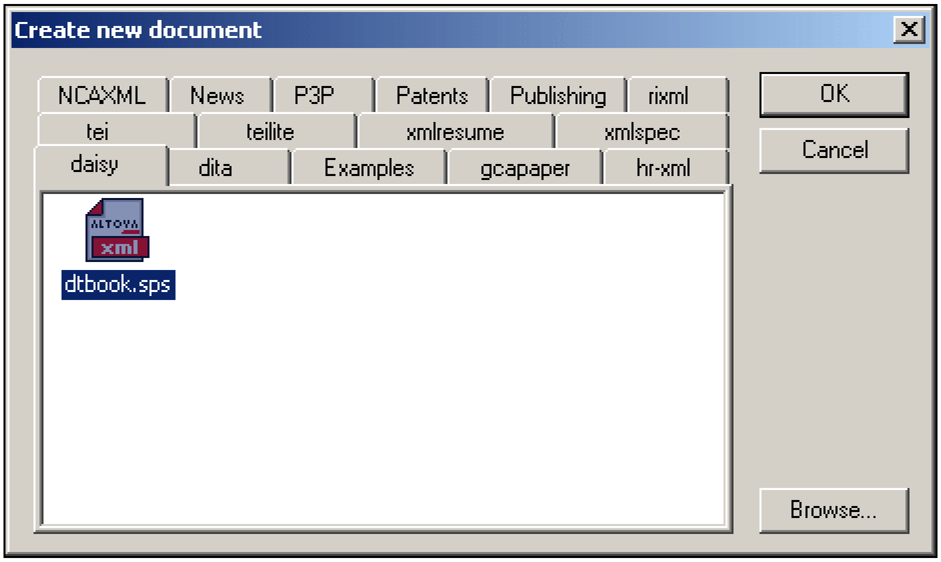
The **Authentic** menu contains commands relevant to editing XML documents in Authentic View . For a tutorial on Authentic View, see the Tutorials section.



11.8.1 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.



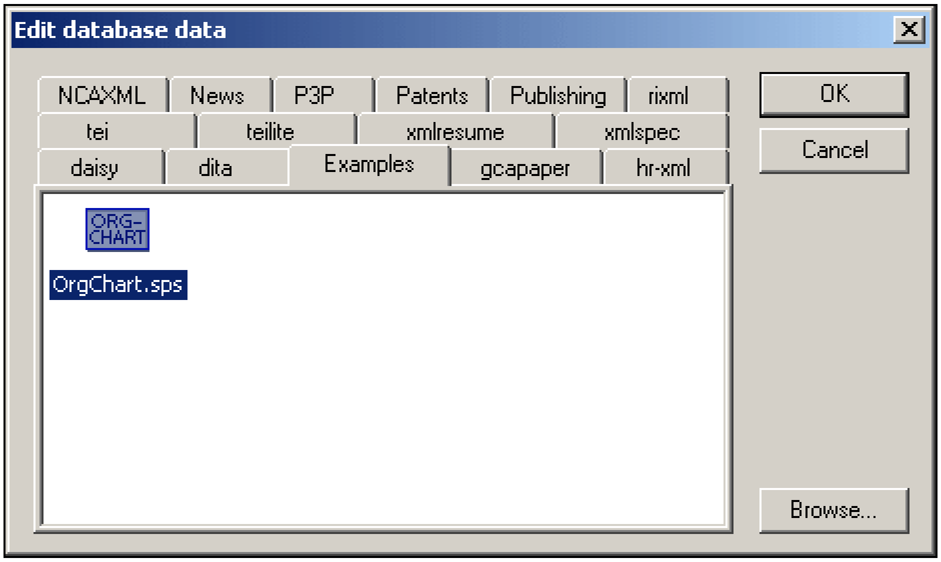
Browse for the required SPS file, and select it. This opens an XML document template in Authentic View.

Please 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.

11.8.2 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.

11.8.3 Assign a StyleVision Stylesheet...

This 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.

```
<?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.

11.8.4 Edit StyleVision Stylesheet

The **Authentic | Edit a StyleVision Stylesheet** command starts StyleVision and allows you to edit the StyleVision Power Stylesheet immediately in StyleVision.

11.8.5 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.

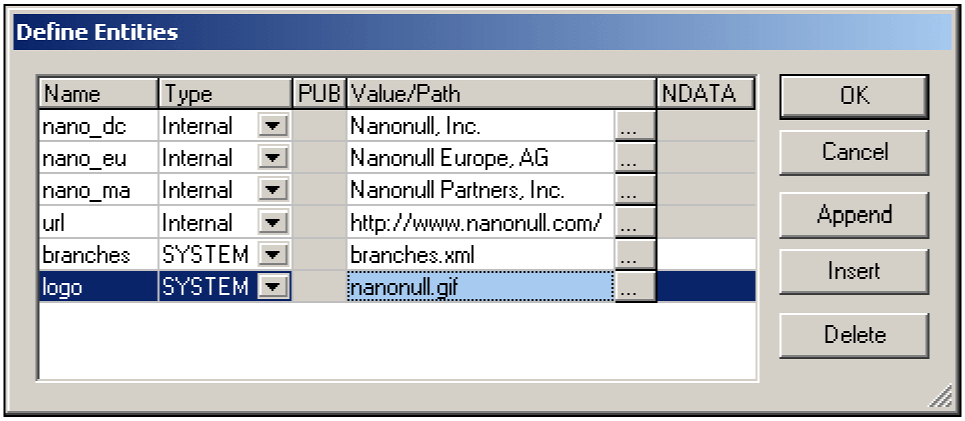
An entity is useful if you will be using a text string, XML fragment, or some other external resource in multiple locations in your document. You define the entity, which is basically a short name that stands in for the required data, in the Define Entities dialog. After defining an entity you can use 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.



Name	Type	PUB	Value/Path	NDATA
nano_dc	Internal		Nanonull, Inc.	
nano_eu	Internal		Nanonull Europe, AG	
nano_ma	Internal		Nanonull Partners, Inc.	
url	Internal		http://www.nanonull.com/	
branches	SYSTEM		branches.xml	
logo	SYSTEM		nanonull.gif	

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. Note that entities are a good mechanism for including Unicode characters in your document; do this by entering the Unicode number as the value of an internal entity.
   - 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.

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 StyleVision Power Stylesheet.

11.8.6 Hide markup

This command hides markup symbols in the Authentic View.

11.8.7 Show Small markup

This command shows small markup symbols in the Authentic View.

11.8.8 Show Large markup

This command shows large markup symbols in the Authentic View.

11.8.9 Show Mixed markup

This 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.

11.8.10 Append row

This command appends a row to the current table in Authentic View.

11.8.11 Insert row

This command inserts a row into the current table in Authentic View.

11.8.12 Duplicate row

This command duplicates the current table row in Authentic View.

11.8.13 Move row Up

This command moves the current table row up by one row in Authentic View.

11.8.14 Move row Down

This command moves the current table row down by one row in Authentic View.

11.8.15 Delete row

This command deletes the current table row in Authentic View.

11.9 Convert Menu

XMLSpy provides powerful data exchange functions that allow you to:

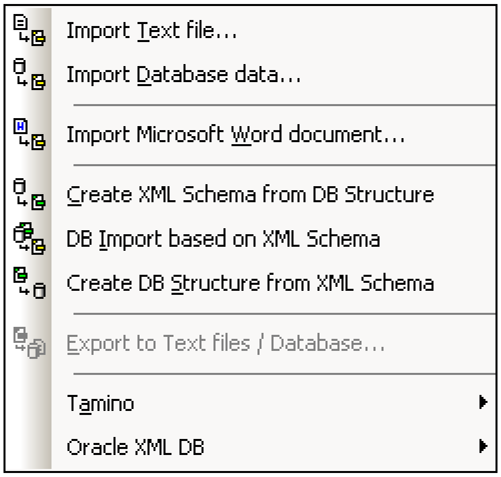
- Import and export text, word processor, database, and XML files.
- Create an XML Schema based on an existing database
- Import database data based on an existing XML Schema
- Create a **database structure**, based on an existing XML schema

These functions are available on the **Convert** menu (*see screenshot*).

XMLSpy currently supports the following databases and connection methods:

- Microsoft Access 2000 and 2003 (ADO)
- Microsoft SQL Server (ADO)
- Oracle (OCI)
- MySQL (ODBC)
- Sybase (ODBC)
- IBM DB2 (ODBC)

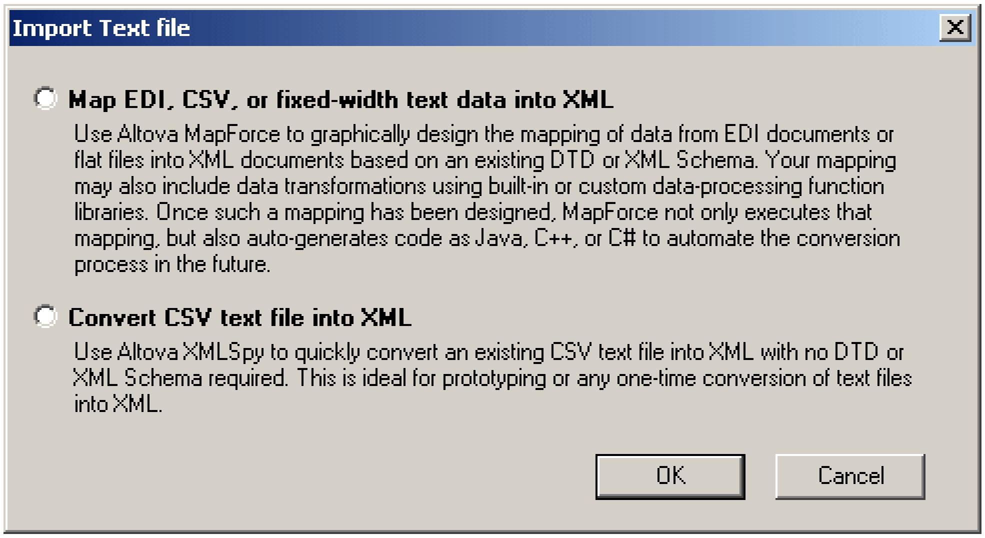
- Any ADO
- Any ODBC


Import Text file...
Import Database data...
Import Microsoft Word document...
Create XML Schema from DB Structure
DB Import based on XML Schema
Create DB Structure from XML Schema
Export to Text files / Database...
Tamino
Oracle XML DB

11.9.1 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, as most software products support a text export interface of some kind.

1. Select the menu item **Convert | Import Text file**. The following dialog appears:



Import Text file

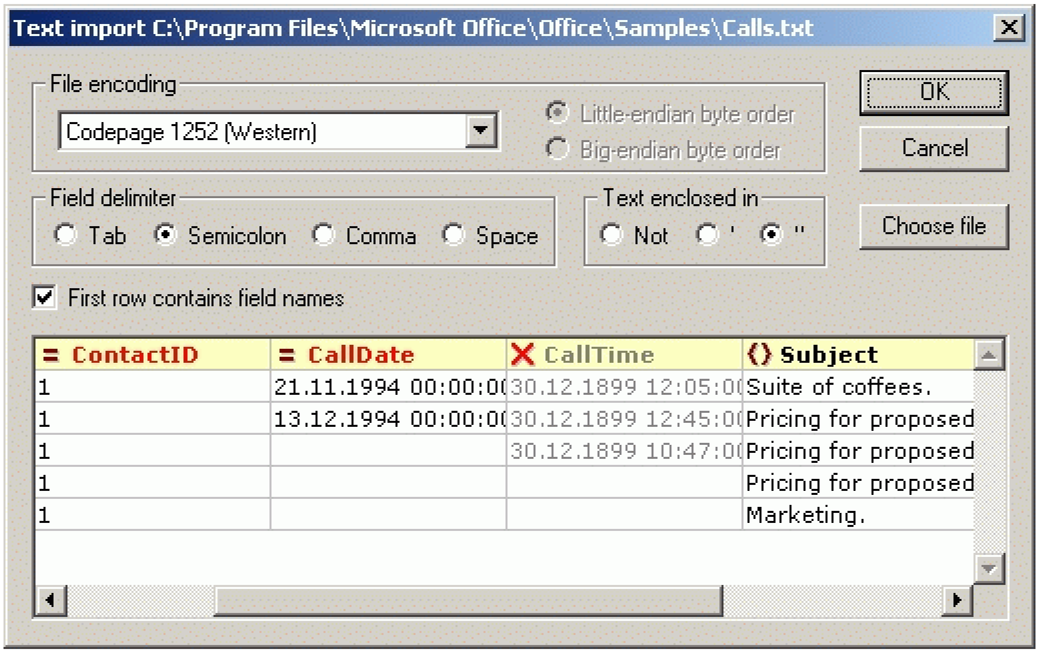
○ **Map EDI, CSV, or fixed-width text data into XML**

Use Altova MapForce to graphically design the mapping of data from EDI documents or flat files into XML documents based on an existing DTD or XML Schema. Your mapping may also include data transformations using built-in or custom data-processing function libraries. Once such a mapping has been designed, MapForce not only executes that mapping, but also auto-generates code as Java, C++, or C# to automate the conversion process in the future.

○ **Convert CSV text file into XML**

Use Altova XMLSpy to quickly convert an existing CSV text file into XML with no DTD or XML Schema required. This is ideal for prototyping or any one-time conversion of text files into XML.

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.



Text import C:\Program Files\Microsoft Office\Office\Samples\Calls.txt

File encoding: Codepage 1252 (Western)

● Little-endian byte order
○ Big-endian byte order

Field delimiter: ○ Tab ● Semicolon ○ Comma ○ Space

Text enclosed in: ○ Not ○ ' ● "

☑ First row contains field names

= ContactID	= CallDate	✕ CallTime	{} Subject
1	21.11.1994 00:00:0(	30.12.1899 12:05:0(	Suite of coffees.
1	13.12.1994 00:00:0(	30.12.1899 12:45:0(	Pricing for proposed
1		30.12.1899 10:47:0(	Pricing for proposed
1			Pricing for proposed
1			Marketing.

File 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. 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.

Field 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).

Text enclosed in
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.

First row contains field names
It is also very common for text files to contain the field names in the first row within the file. XMLSpy allows you to define your XML element or attribute names according to this information.

Choose file
Click on the **Choose file** button to select the specific file to be imported, after having defined the import parameters. The file name will be retained the next time you use this dialog box, allowing you to change settings and re-import the same file if the resulting XML file does not match your expectations.

Having selected the file to be imported, you are provided with a preview of the data import. Any changes in the above options will be reflected in the preview immediately.

Renaming field or column names
Rename a field or column name by clicking on its title and editing the name.

Attribute, Element, or no columns
XMLSpy lets you choose if you want to import a column as an attribute, element, or if you'd rather skip the column entirely.

Click on the icons to the left of the column titles to toggle between these three options. In the example above,
ContactID and CallDate - are imported as attributes
CallTime - is not imported
Subject - is imported as an element.

11.9.2 Import Database data...

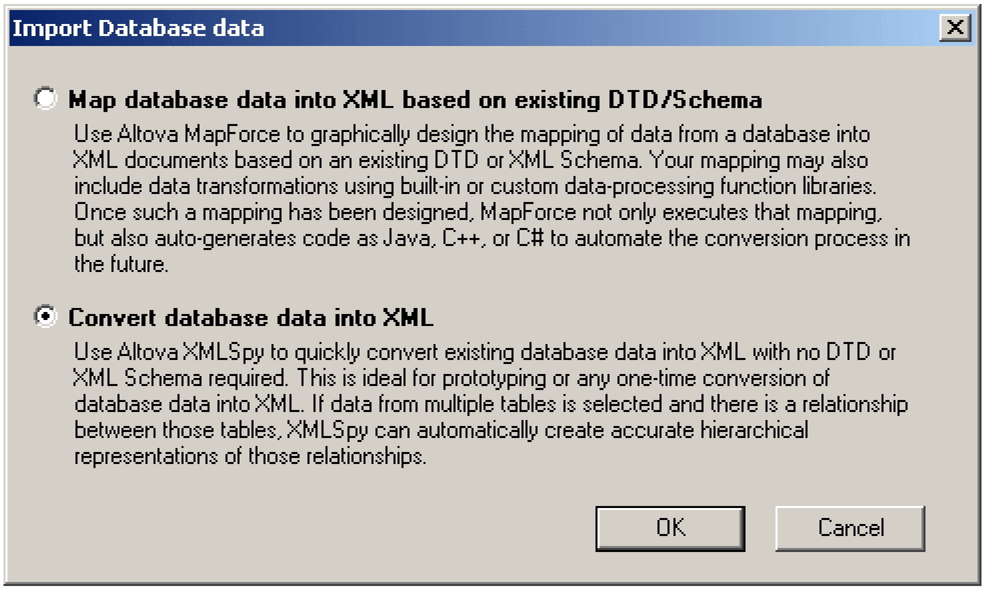
This command lets you import database data from many sources, with the restriction that only **one table** can be imported at a time.

The method described in the following text is (practically) identical in all of the following **Convert** menu options, and for all types of database connections:

- Import Database data
- Create XML Schema from DB Structure
- DB Import based on XML Schema
- Create DB Structure from XML Schema

To import database data:

1. Select the menu option **Convert | Import Database data**. The following dialog appears:



Import Database data

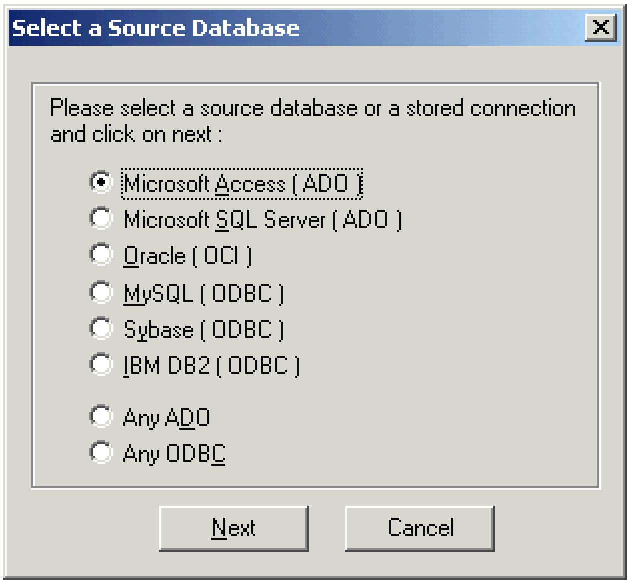
○ **Map database data into XML based on existing DTD/Schema**

Use Altova MapForce to graphically design the mapping of data from a database into XML documents based on an existing DTD or XML Schema. Your mapping may also include data transformations using built-in or custom data-processing function libraries. Once such a mapping has been designed, MapForce not only executes that mapping, but also auto-generates code as Java, C++, or C# to automate the conversion process in the future.

● **Convert database data into XML**

Use Altova XMLSpy to quickly convert existing database data into XML with no DTD or XML Schema required. This is ideal for prototyping or any one-time conversion of database data into XML. If data from multiple tables is selected and there is a relationship between those tables, XMLSpy can automatically create accurate hierarchical representations of those relationships.

2. Select one of the following:
   - **Map database data into XML based on existing DTD/Schema** (you must have installed Altova MapForce in order to select this option)
   - **Convert database data into XML**
3. Click **OK**. The Select a Source Database dialog box opens.



Select a Source Database

Please select a source database or a stored connection and click on next :

● Microsoft Access (ADO)
○ Microsoft SQL Server (ADO)
○ Oracle (OCI)
○ MySQL (ODBC)
○ Sybase (ODBC)
○ IBM DB2 (ODBC)
○ Any ADO
○ Any ODBC

4. Select the database you want to use as a data source (MS Access, for example), and click **Next** to create the connection to the selected database.
5. Click the link of one the databases listed below to see how the sequence continues for that specific database:

   MS Access (ADO)
   MS SQL Server (ADO)
   Oracle (OCI)
   IBM DB2 (ODBC)

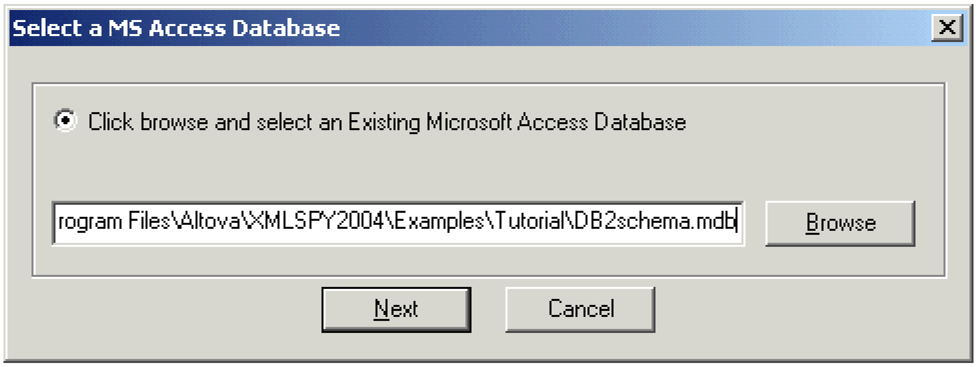
6. Click **Import** to import the database data.

Please note:
All connection sequences using ODBC are identical: MySQL, Sybase and IBM DB2.

MS Access as source

XMLSpy supports MS Access 2000 and MS Access 2003 databases.

1. Having selected **Microsoft Access** and clicked **Next**, click the **Browse** button to select the respective database, then click **Open** to select and return to this dialog box.
   The path and name of the database are now visible in the text box.

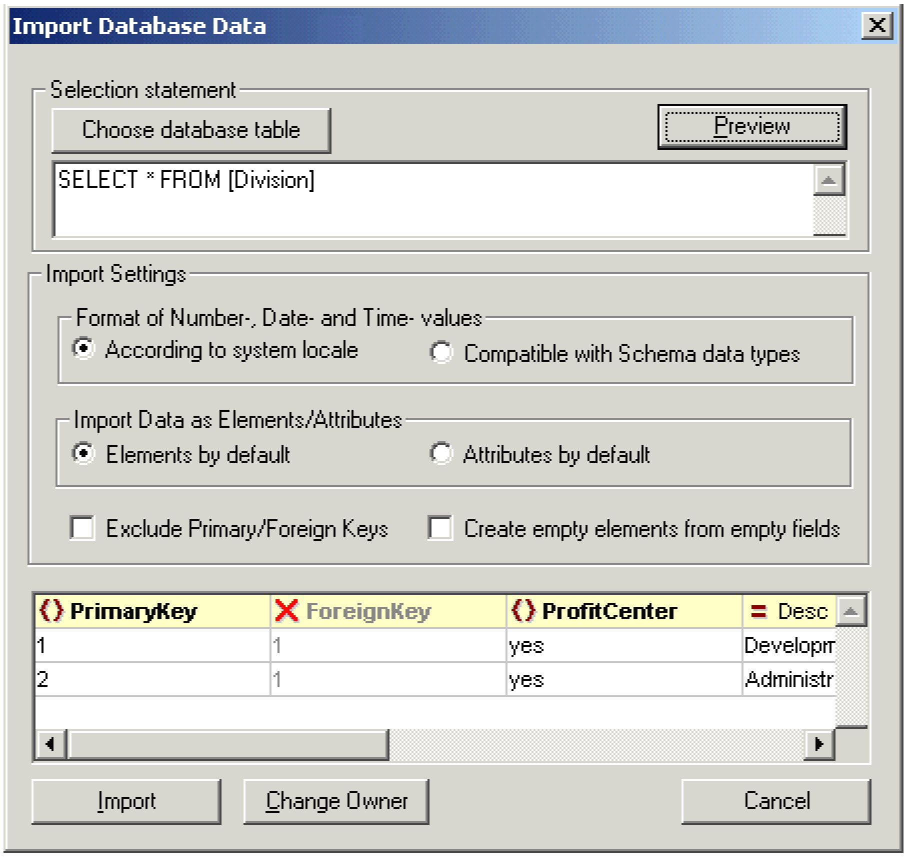


Select a MS Access Database

● Click browse and select an Existing Microsoft Access Database

rogram Files\Altova\XMLSPY2004\Examples\Tutorial\DB2schema.mdb

2. Click **Next** to continue.
   This opens the Import Database Data dialog box.



Import Database Data

Selection statement

SELECT * FROM [Division]

Import Settings

Format of Number-, Date- and Time- values
● According to system locale ○ Compatible with Schema data types

Import Data as Elements/Attributes
● Elements by default ○ Attributes by default

□ Exclude Primary/Foreign Keys □ Create empty elements from empty fields

() PrimaryKey	✗ ForeignKey	() ProfitCenter	= Desc
1	1	yes	Developm
2	1	yes	Administr

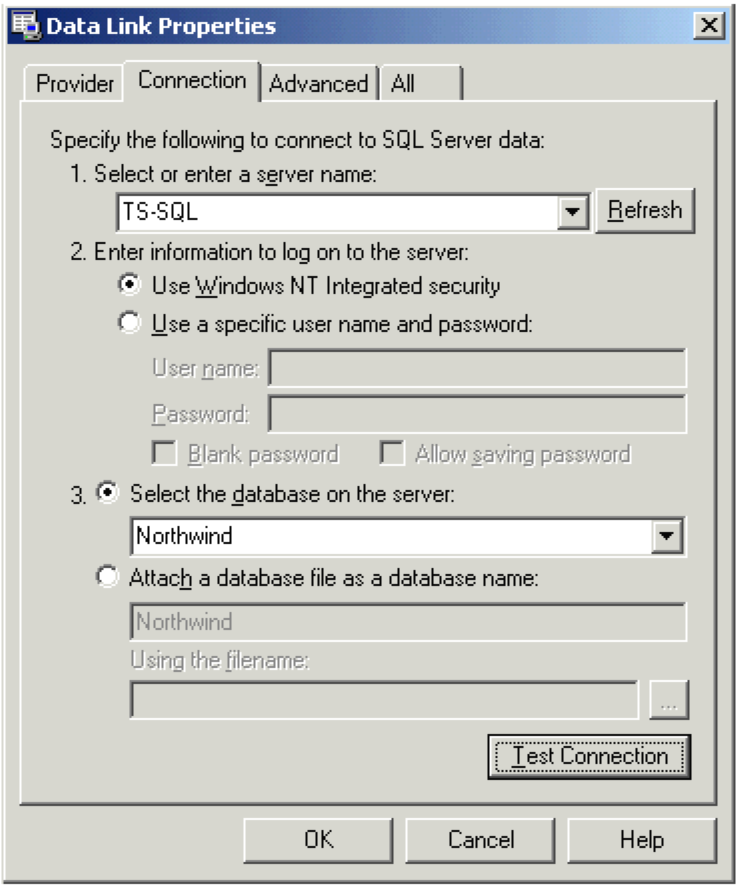
3. Click the **Choose database table** button and select the specific table you want to import (E.g. Division).

The select statement appears in the text box.

4. Click the **Preview** button to preview the database fields.
   You can now further define the import settings by selecting the specific options from the Import Settings group
5. Click the **Import** button to import the database data.

MS SQL as source

1. Having selected "**Microsoft SQL Server**" and clicked **Next**.
   The "MS SQL Server : ADO connection String" dialog box is opened.
2. Click the **Build** button to start the process of defining an ADO connection string.
   This opens the Data Link Properties dialog box, where you specify the connection server, login properties, etc.
3. Fill in the necessary fields and click the **Test Connection** button.



Specify the following to connect to SQL Server data:

1. Select or enter a server name: TS-SQL

2. Enter information to log on to the server:

● Use Windows NT Integrated security

○ Use a specific user name and password:

User name:

Password:

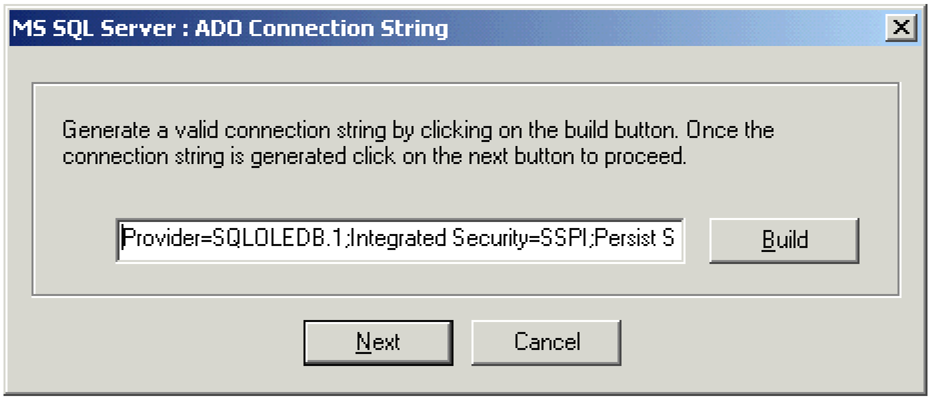
□ Blank password □ Allow saving password

3. ● Select the database on the server: Northwind

○ Attach a database file as a database name: Northwind

Using the filename:

4. Click **OK** to complete the connection string definition.
   The connection string appears in the text box.

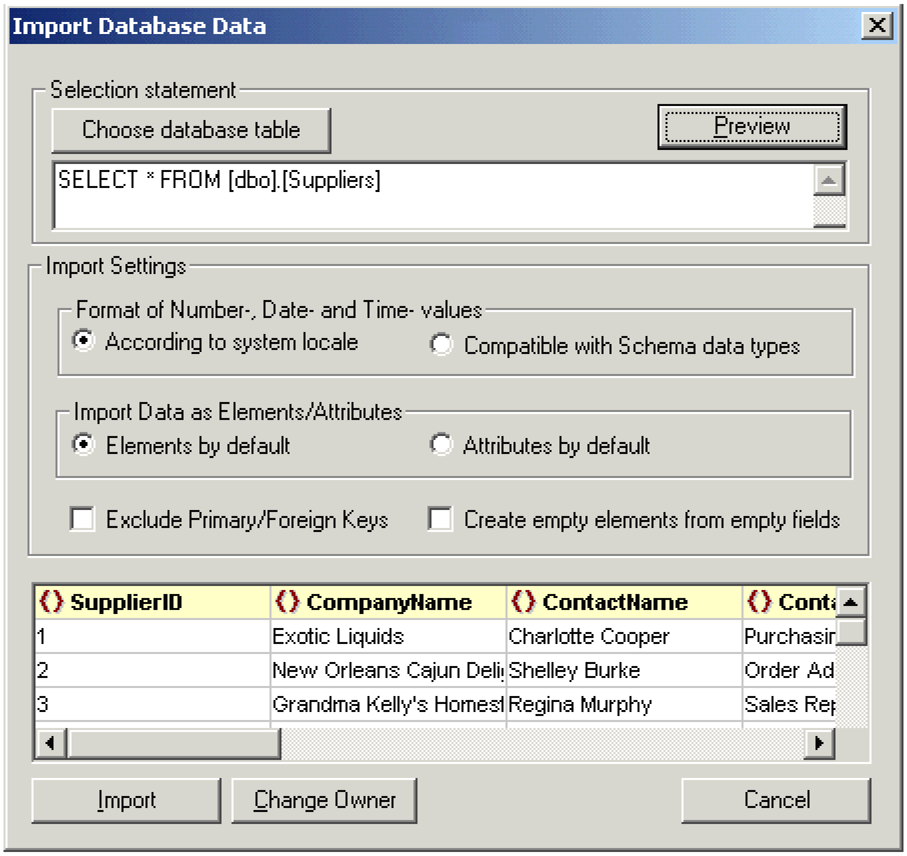


MS SQL Server : ADO Connection String

Generate a valid connection string by clicking on the build button. Once the connection string is generated click on the next button to proceed.

Provider=SQLOLEDB.1;Integrated Security=SSPI;Persist S

5. Click **Next** to continue.
   This opens the Import Database Data dialog box
6. Click the **Choose database table** button to select the specific table you want to import e.g. suppliers.
   The select statement appears in the text box.
7. Click the **Preview** button to preview the database fields.



Import Database Data

Selection statement

SELECT * FROM [dbo].[Suppliers]

Import Settings

Format of Number-, Date- and Time- values

● According to system locale ○ Compatible with Schema data types

Import Data as Elements/Attributes

● Elements by default ○ Attributes by default

□ Exclude Primary/Foreign Keys □ Create empty elements from empty fields

SupplierID	CompanyName	ContactName	Conta
1	Exotic Liquids	Charlotte Cooper	Purchasir
2	New Orleans Cajun Deli	Shelley Burke	Order Ad
3	Grandma Kelly's Homest	Regina Murphy	Sales Rep

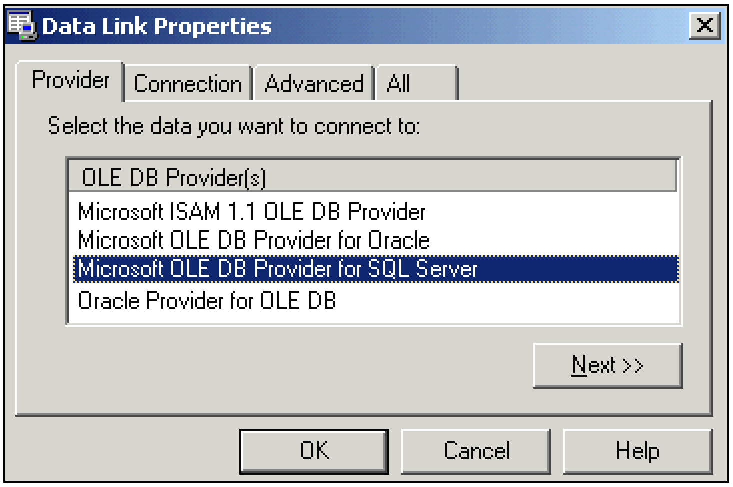
You can now further define the import settings by selecting the specific options from the Import Settings group

8. Click the **Import** button to import the database data.

ADO connection string

If you need help in entering an ADO connection string, click the **Build** button in the ADO Connection String dialog, which opens the ADO Data Link properties dialog box.

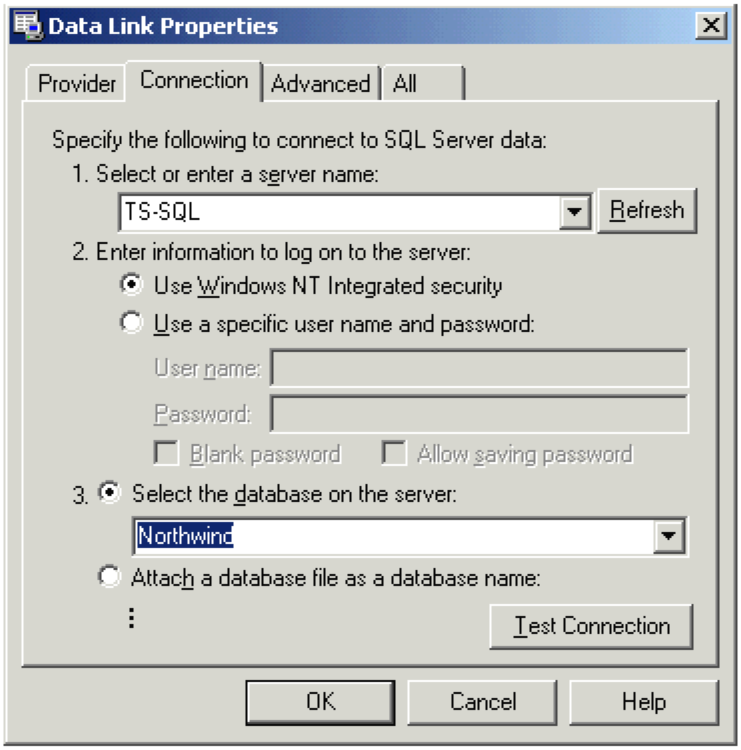
ADO Data Link properties dialog box:



Select the data you want to connect to:

OLE DB Provider(s)
Microsoft ISAM 1.1 OLE DB Provider
Microsoft OLE DB Provider for Oracle
Microsoft OLE DB Provider for SQL Server
Oracle Provider for OLE DB

Select the corresponding OLE DB Provider from the list and click **Next** to enter the connection properties:



Specify the following to connect to SQL Server data:
1. Select or enter a server name: TS-SQL
2. Enter information to log on to the server:
● Use Windows NT Integrated security
○ Use a specific user name and password:
User name:
Password:
□ Blank password □ Allow saving password
3. ● Select the database on the server: Northwind
○ Attach a database file as a database name:
⋮

XMLSpy relies on the ActiveX Data Object (ADO) interface for much of the basic database connectivity. ADO is included with Windows 2000, Windows XP and Microsoft Office 2000, but you may need to install it if you are using an older version of Windows or Office.

Please note:

Activate the **Allow saving password** check box, to enter your password info into the connection string, and avoid an error message when the connection takes place.

Please see our FAQ for more information on ADO and to download the Microsoft Data Access

Components (MDAC) which allow you to upgrade to the latest ADO version. We also recommend that you visit the Microsoft Universal Data Access site on the Internet.

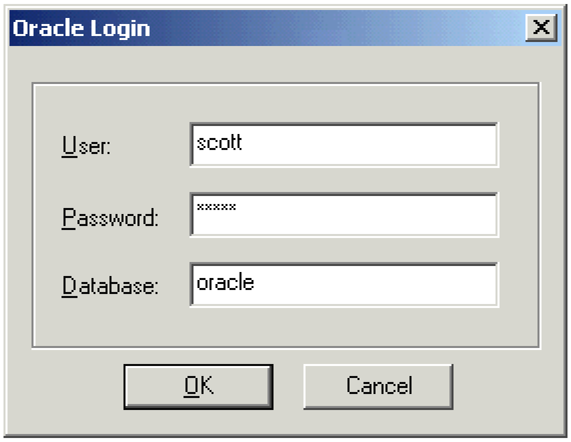
After you have imported your data and converted it to XML format, you may also want to use the Generate DTD/Schema function to get all the datatype information you need from your database into your XML documents.

XMLSpy now makes it possible for you to convert a database into a schema. MS Access and several other databases are able to automatically provide the key and keyref information for the ADO driver, used to create the database hierarchy.

Please see the section under the **Convert menu | Create XML Schema from DB Structure** for more information.

Oracle as source

1. Having selected "**Oracle (OCI)**" and clicked **Next**.
   The "**Oracle Login**" dialog box is opened.
2. Enter the Login parameters and click **OK**.



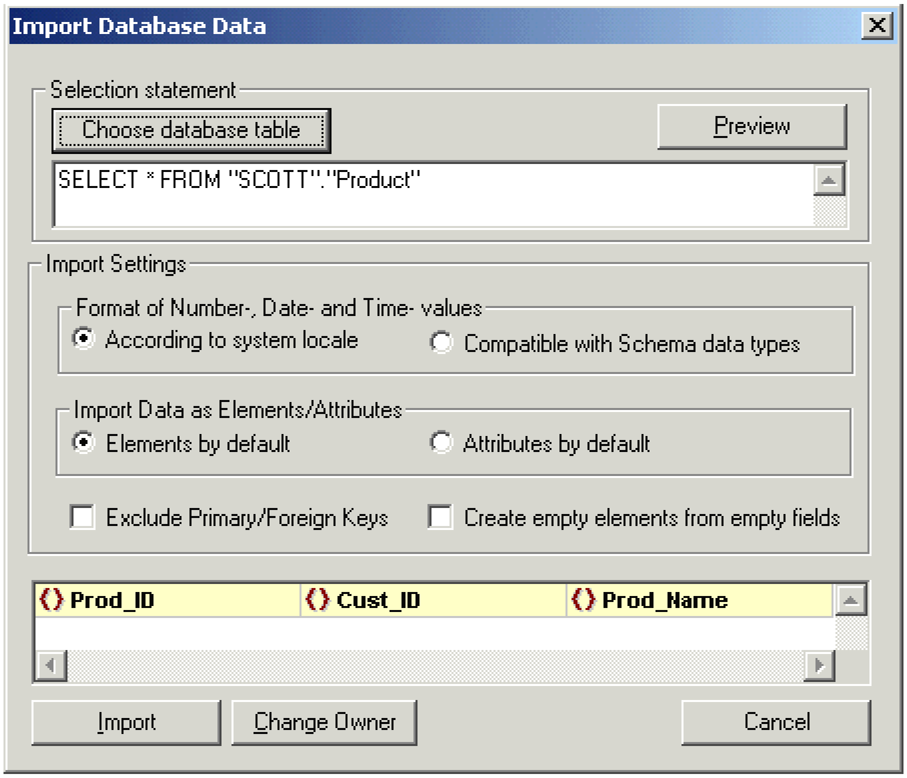
Oracle Login

User: scott

Password: *****

Database: oracle

   This opens the **Import Database Data** dialog box.
3. Click the **Choose database table** button to select the specific table you want to import. The select statement appears in the text box.
4. Click the **Preview** button to preview the database fields.



Selection statement

SELECT * FROM "SCOTT"."Product"

Import Settings

Format of Number-, Date- and Time- values

● According to system locale ○ Compatible with Schema data types

Import Data as Elements/Attributes

● Elements by default ○ Attributes by default

□ Exclude Primary/Foreign Keys □ Create empty elements from empty fields

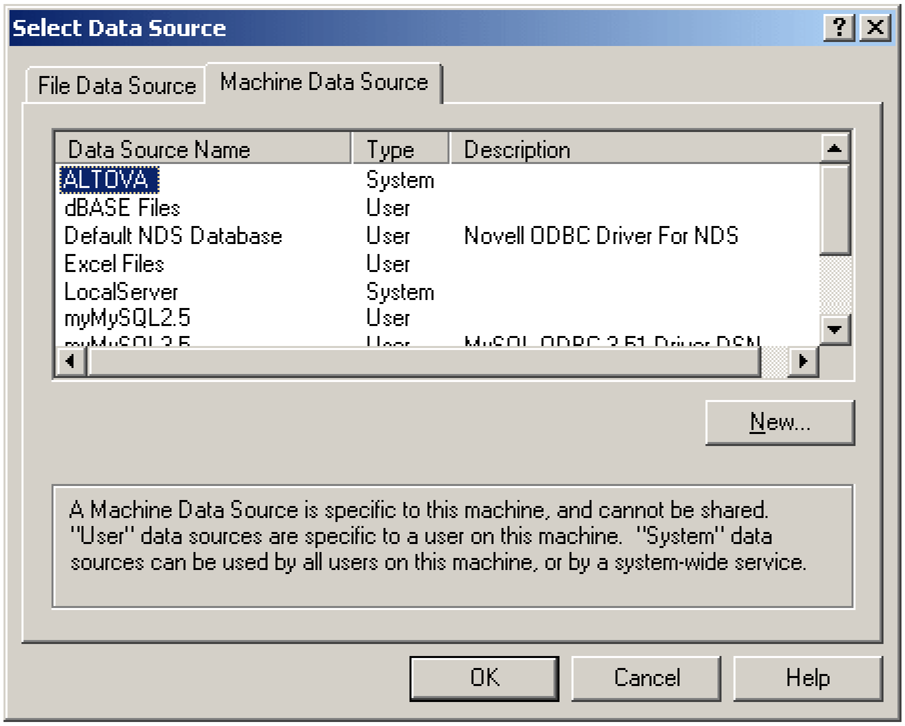
Prod_ID	Cust_ID	Prod_Name

You can now further define the import settings by selecting the specific options from the Import Settings group

5. Click the **Import** button to import the database data.

IBM DB2 as source

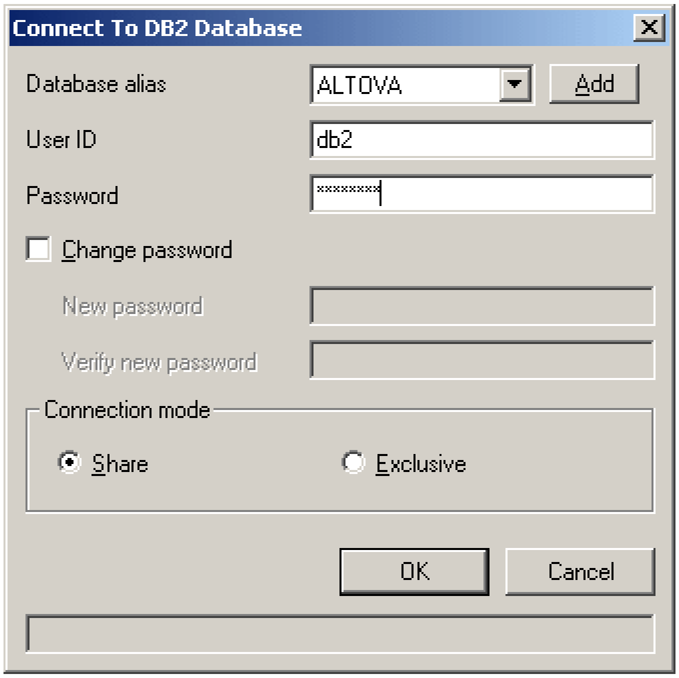
1. Having selected "**IBM DB2 (ODBC)**" and clicked **Next**.
   The "**IBM DB2 : ODBC Connection String**" dialog box is opened.
2. Click the **Build** button to start the process of defining an ODBC connection string.
   This opens the "Select Data Source" dialog box.



Data Source Name	Type	Description
ALTOVA	System	
dBASE Files	User	
Default NDS Database	User	Novell ODBC Driver For NDS
Excel Files	User	
LocalServer	System	
myMySQL2.5	User	
myMySQL3.5	User	MySQL ODBC 3.51 Driver DSN

A Machine Data Source is specific to this machine, and cannot be shared. "User" data sources are specific to a user on this machine. "System" data sources can be used by all users on this machine, or by a system-wide service.

3. Click the **Machine Data Source** tab, select the data source name created when you installed the IBM DB2 client, and click **OK**.
   The "Connect to DB2" dialog box is opened.
4. Enter the **User ID** and **password**, and click **OK** to connect to the database.



Database alias: ALTOVA

User ID: db2

Password: ********

☐ Change password

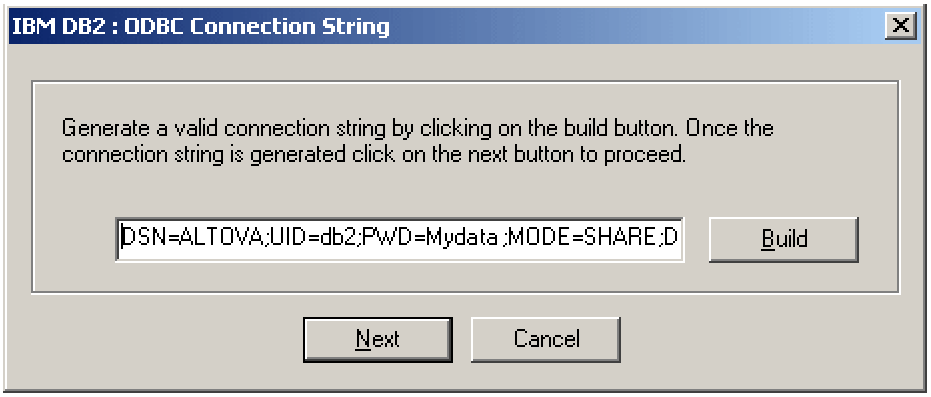
New password

Verify new password

Connection mode

● Share ○ Exclusive

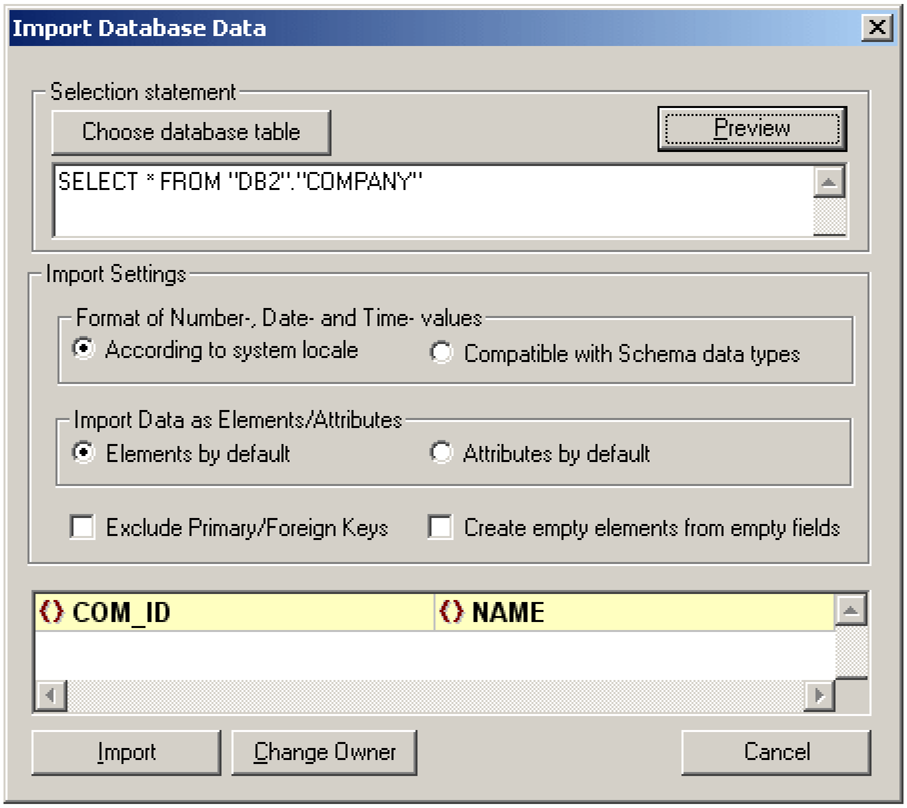
The connection string is created and placed in the connection string text box.



Generate a valid connection string by clicking on the build button. Once the connection string is generated click on the next button to proceed.

DSN=ALTOVA;UID=db2;PWD=Mydata ;MODE=SHARE;D

5. Click **Next** to continue.
   This opens the "**Import Database Data**" dialog box.
6. Click the **Choose database table** button to select the specific table you want to import. The select statement appears in the text box.
7. Click the **Preview** button to preview the database fields.



SELECT * FROM "DB2"."COMPANY"

Format of Number-, Date- and Time- values
● According to system locale ○ Compatible with Schema data types

Import Data as Elements/Attributes
● Elements by default ○ Attributes by default

□ Exclude Primary/Foreign Keys □ Create empty elements from empty fields

COM_ID	NAME

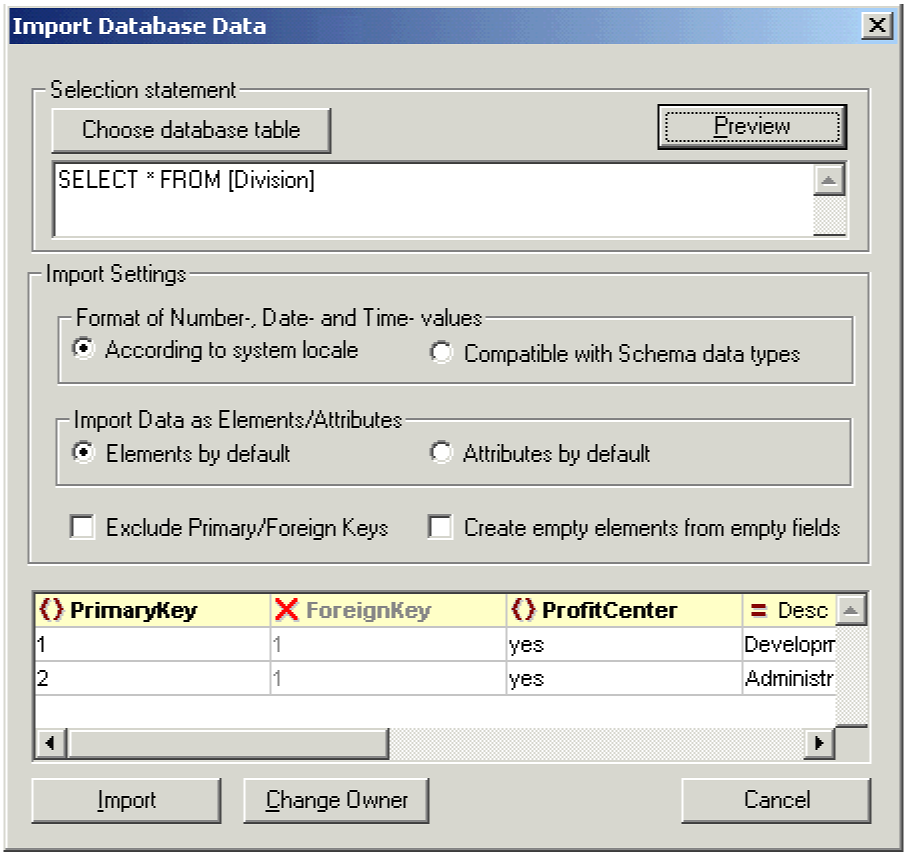
You can now further define the import settings by selecting the specific options from the Import Settings group

8. Click the **Import** button to import the database data.

Database import settings

Selection statement

Click on the "Choose database table" button, to select the database, or enter an arbitrary SELECT statement to create the record-set you intend to import.



Selection statement

SELECT * FROM [Division]

Import Settings

Format of Number-, Date- and Time- values: ● According to system locale ○ Compatible with Schema data types

Import Data as Elements/Attributes: ● Elements by default ○ Attributes by default

□ Exclude Primary/Foreign Keys □ Create empty elements from empty fields

PrimaryKey	ForeignKey	ProfitCenter	Desc
1	1	yes	Developm
2	1	yes	Administr

Having selected the data source, click the **Preview** button to verify the data that you wish to import.

Please note:

The automatically generated select statement **can be edited** if you want manual control over the import process.

Format of Number, Date and Time values

XMLSpy lets you choose different representations for date and number formats - depending on whether you intend to use the resultant XML file in conjunction with the new unified datatypes proposed by the most recent XML Schema draft, or if you want to keep those formats corresponding to the locale in use in your country.

Import data as Elements/Attributes

These options let you specify how you want the parent and child elements to be imported: as elements or attributes. The preview is updated when you select one of these options.

Exclude Primary/Foreign keys

This option enables you to exclude the primary or foreign keys for all the tables you are importing.

Create empty elements from empty fields

This option lets you create empty elements for all the empty fields that exist in the tables you import.

Attribute, Element, or no columns

XMLSpy lets you choose if you want to import a parent column as an attribute, element, or if you'd rather skip the column entirely.

Click on the icons to the left of the **column titles**, in the preview window, to toggle between these three options. In the example above,

PrimaryKey and ProfitCenter - are imported as elements
ForeignKey - is not imported
Desc - is imported as an attribute.

Please note:

The selections you make here initially apply to the parent items as well. Changing parent items directly, takes precedence over these settings, but only applies to the parent elements and not to any child elements. Clicking the **Preview** button, resets any changes made to the parent elements in the Preview view.

Defining which fields to import:

The preview window allows you to directly select and define the field data you want to import.

Clicking repeatedly on the **element symbol** ⟨⟩ to the left of the element name, cycles through the available possibilities:

⟨⟩	Define and import this field as an **Element**.
=	Define and import this field as an **Attribute**.
☒	**Skip**, do not import this field.

Change Owner

This option presents a list of owners available to the user logged onto the database. Use this option when you wish to execute the SQL command for a different user.

11.9.3 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.

11.9.4 Create XML Schema from DB Structure

XMLSpy enables you to create a schema based upon an external database file. The following databases are supported:

- Microsoft Access (2000 and 2003)
- Microsoft SQL Server
- Oracle
- MySQL
- Sybase
- IBM DB2
- ADO compatible databases

- ODBC databases

XMLSpy also allows you to create two different types of database schema, hierarchical or flat.

- 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.
- 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.

  For more information about schema formats, schema **extensions**, tables and **identity constraints** please see the Create Database options.

Please note:
Key and Keyref entries are automatically generated by the "Create DB Structure from XML Schema" for the major databases.

The table below shows the type of database created, the restrictions, and the connecting methods, when using the "Create XML Schema from DB Structure" menu option.

	Create XML Schema from DB Structure			
	Directly	using ODBC (unique keys are not supported by ODBC)	using ADO	Flat, or hierarchical DB format
Microsoft Access	OK *****	OK (not recommended) Primary and Foreign keys are not supported.	OK	hierarchical
MS SQL Server	OK *****	OK	OK	flat and hierarchical
Oracle	OK *****	OK, restrictions: table containing columns of type CLOB, BLOB, BFILE; XML tables	OK, restrictions: table containing columns of type CLOB, BLOB, BFILE; XML tables; owner information, Identity constraints are not read from the database	flat and hierarchical
MySQL	-	OK *****	OK ⌖	hierarchical
Sybase	-	OK *****	OK	flat and hierarchical
IBM DB2	-	OK *****	OK	flat and hierarchical

***** **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**

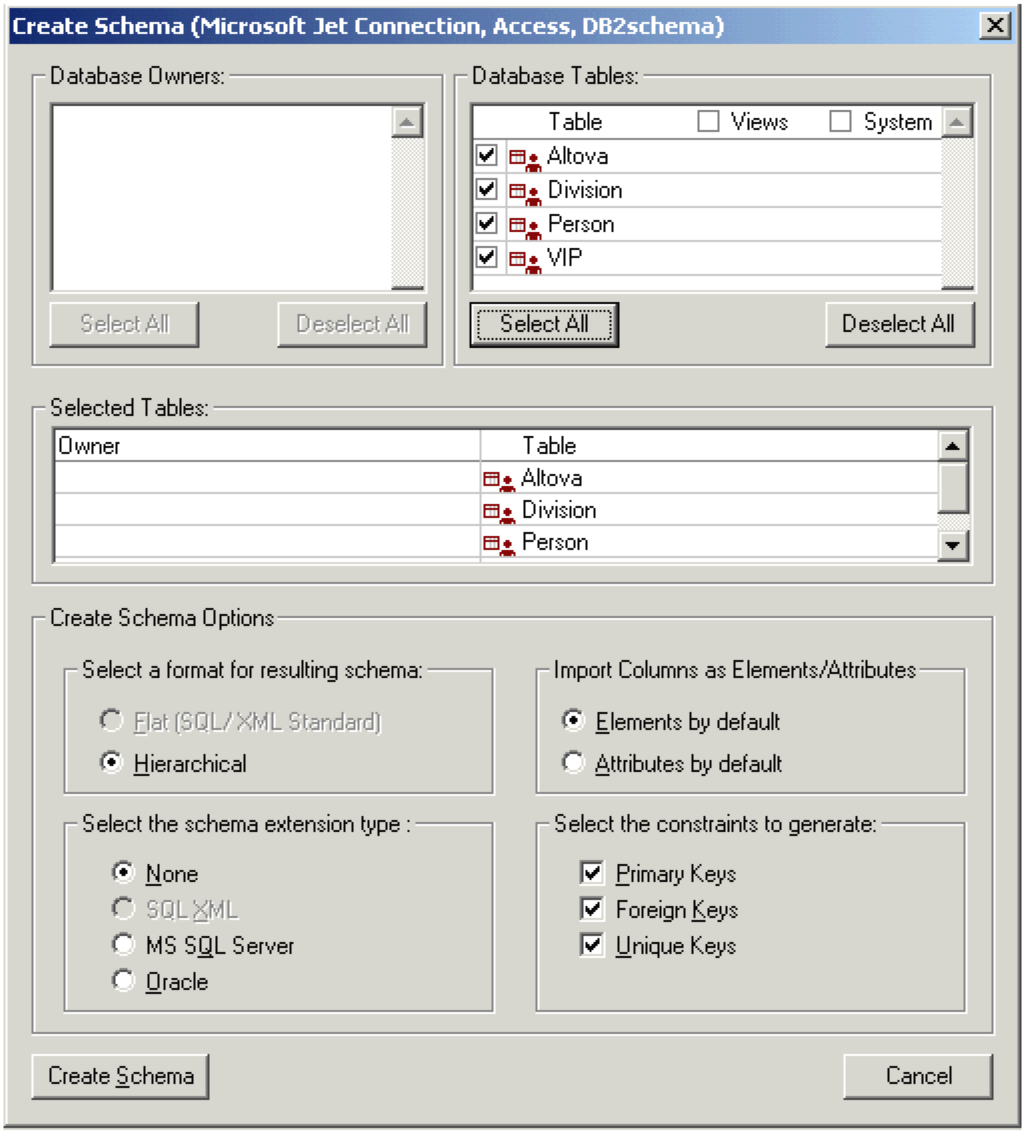
Please also see Mapping to XML Schema datatypes for specific information on how database specific datatypes are mapped to individual schema datatypes.

Create XML Schema from DB Structure (MS Access)

XMLSpy supports MS Access 2000 and MS Access 2003 databases.

To create a hierarchical schema from a database file (MS Access):

1. Select the menu option **Convert | Create XML Schema from DB Structure.**
2. Select **Microsoft Access**, and click **Next**.
3. Click the **Browse** button and select the MS Access source database, for example, the **DB2schema.mdb** file supplied with XMLSpy (in the `Tutorial` folder), and click the **Open** button.
4. Click **Next** to continue.
   This opens the **Create Schema** dialog box, which allows you to define specific database tables and other schema settings.



Create Schema (Microsoft Jet Connection, Access, DB2schema)

Database Owners:

Select All Deselect All

Database Tables:

Table	□ Views	□ System
☑ Altova		
☑ Division		
☑ Person		
☑ VIP		

Select All Deselect All

Selected Tables:

Owner	Table
	Altova
	Division
	Person

Create Schema Options

Select a format for resulting schema:
○ Flat (SQL/XML Standard)
● Hierarchical

Import Columns as Elements/Attributes
● Elements by default
○ Attributes by default

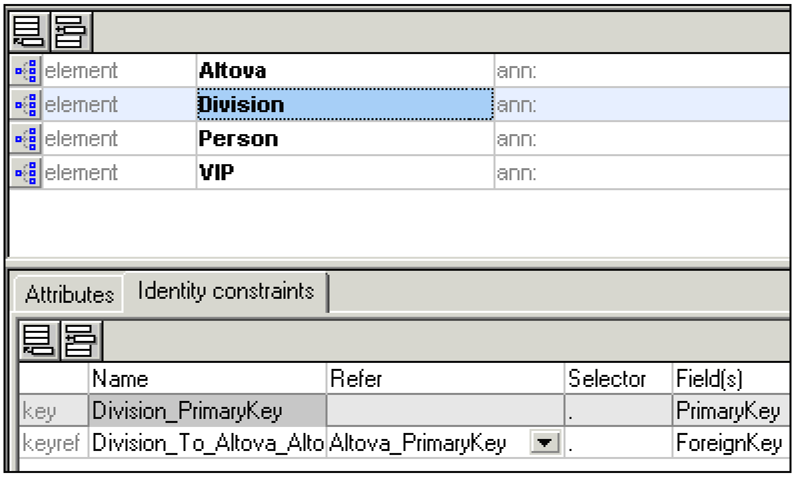
Select the schema extension type :
● None
○ SQL XML
○ MS SQL Server
○ Oracle

Select the constraints to generate:
☑ Primary Keys
☑ Foreign Keys
☑ Unique Keys

Create Schema Cancel

5. Click the **Hierarchical** radio button.
6. Click the **Select All** button.
7. Click the **Create Schema** button to start the conversion process.
   The generated schema appears in the Schema/WSDL Design View. Click the **Identity**

constraints tab, to see the keyref and key fields of the respective elements.

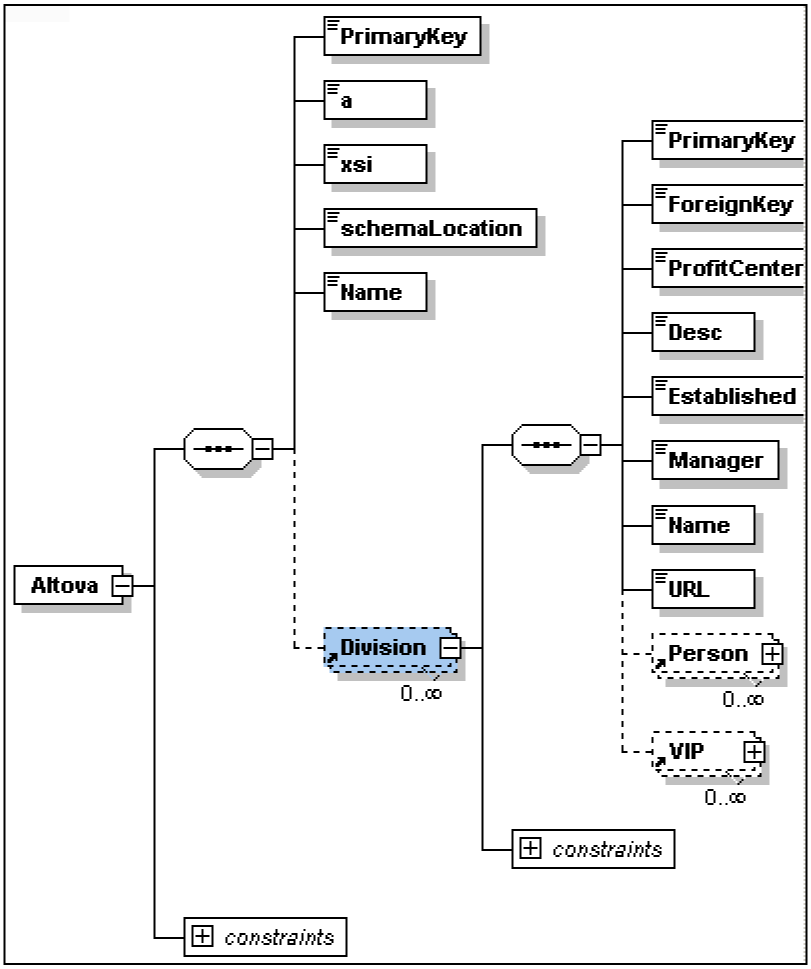


element	**Altova**	ann:
element	**Division**	ann:
element	**Person**	ann:
element	**VIP**	ann:

Attributes | Identity constraints

	Name	Refer	Selector	Field(s)
key	Division_PrimaryKey		.	PrimaryKey
keyref	Division_To_Altova_Alto	Altova_PrimaryKey	.	ForeignKey

8. Click the component icon next to the **Altova** global element, to see the content model.



9. Select the menu option **File | Save as**, and save the new schema e.g. **DB2schema.xsd**.

10. Click the Display all globals icon ▤, to return to the schema overview.

Please note:

When generating the schema, all namespace prefix colons are automatically converted into underscore characters.

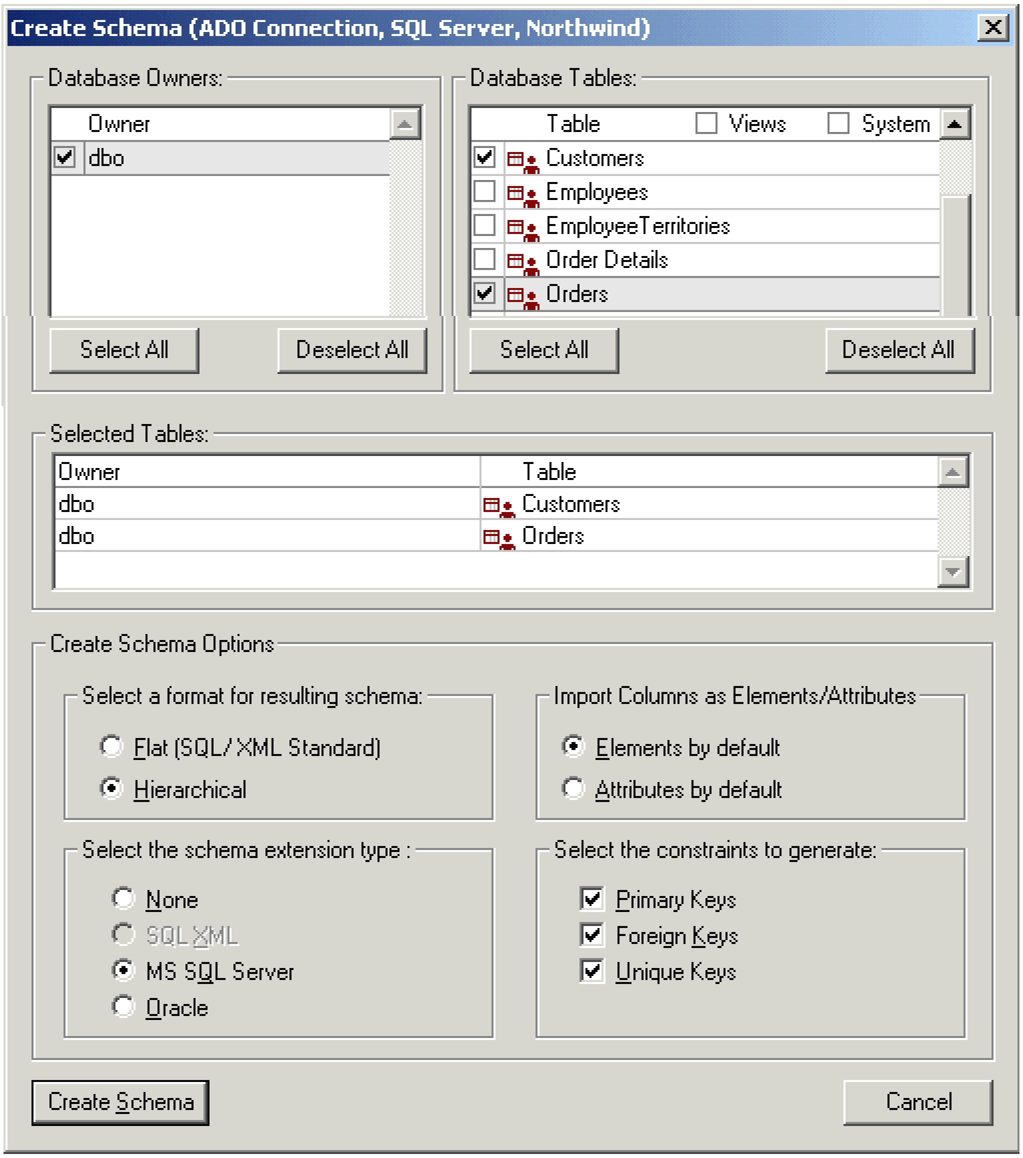
Databases without **Owner** information, such as MS Access, cannot produce flat format databases.

Create XML Schema from DB Structure (Microsoft SQL)

To create a hierarchical schema from a database file (MS SQL):

1. Select the menu option **Convert | Create XML Schema from DB Structure.**
2. Select **Microsoft SQL Server**, and click the **Next** button.
3. Click the **Build** button to select the database connection settings.
   This opens the **Data Link Properties** dialog box.
4. Click the **Microsoft OLE DB Provider for SQL Server** entry (in the Provider tab if necessary), and click **Next**.
5. Fill in the data needed to connect to your SQL database, in the Connection tab, and click **OK** when finished.
   This inserts the **ADO connection string** into the dialog box.
6. Click **Next** to continue.
   This opens the Create Schema dialog box.
7. Click the **Hierarchical** radio button.
8. Click the **Owner** entry in the Database Owner list box (**dbo** in this example), and select the tables you want to appear in the schema (**Customers** and **Orders** in this case).

   Please note: The MS SQL Server Extensions are available in the **SQL** tab in the Details entry helper.



Create Schema (ADO Connection, SQL Server, Northwind)

Database Owners:

	Owner
☑	dbo

Database Tables: □ Views □ System

	Table
☑	Customers
□	Employees
□	EmployeeTerritories
□	Order Details
☑	Orders

Selected Tables:

Owner	Table
dbo	Customers
dbo	Orders

Create Schema Options

Select a format for resulting schema:

○ Flat (SQL/XML Standard)
● Hierarchical

Import Columns as Elements/Attributes

● Elements by default
○ Attributes by default

Select the schema extension type :

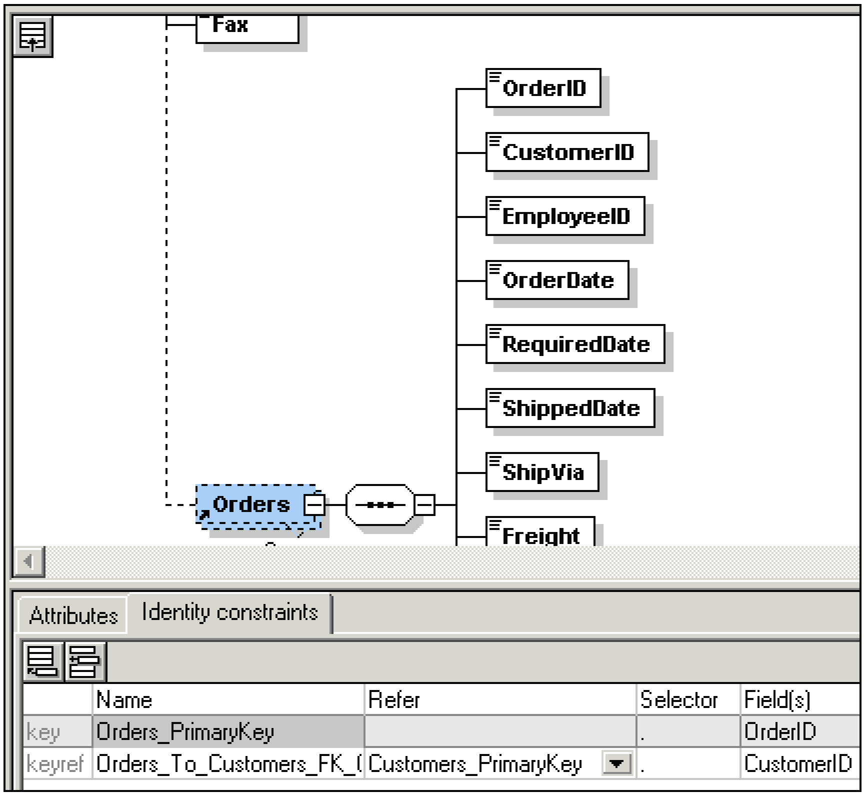
○ None
○ SQL XML
● MS SQL Server
○ Oracle

Select the constraints to generate:

☑ Primary Keys
☑ Foreign Keys
☑ Unique Keys

9. Click the **Create Schema** button to start the conversion process.
   The generated schema appears in the Schema/WSDL View.

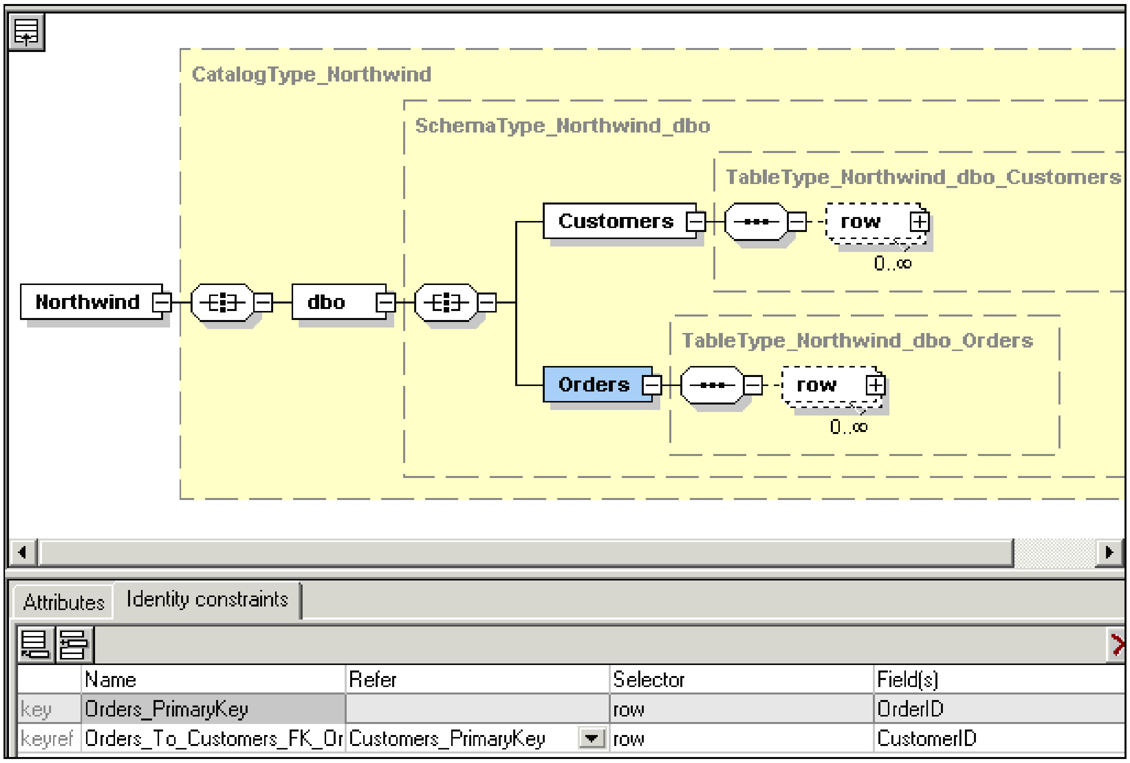
10. Click the component icon next to the **Customers** element to see the content model. Click the **Identity constraints** tab, to see the keyref and key fields of the respective elements. **Note:** the SQL Server Extensions are available in the SQL Server tab, in the Details entry helper.



	Name	Refer	Selector	Field(s)
key	Orders_PrimaryKey		.	OrderID
keyref	Orders_To_Customers_FK_(	Customers_PrimaryKey	.	CustomerID

Result when generating a flat schema model:

If you select the **Flat (SQL/XML Standard)** option in the Create Schema dialog box, the resulting schema appears as shown below.



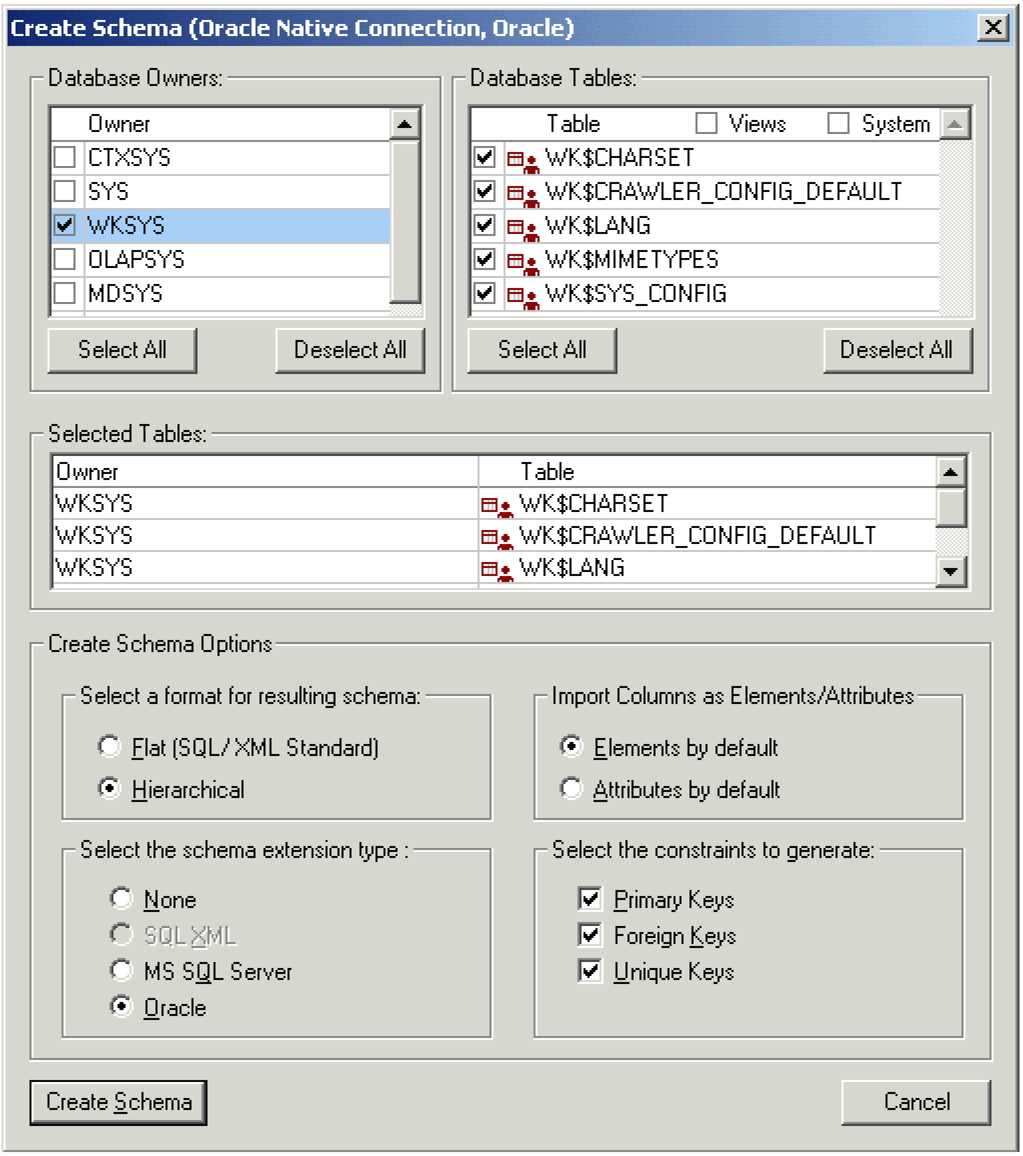
	Name	Refer	Selector	Field(s)
key	Orders_PrimaryKey		row	OrderID
keyref	Orders_To_Customers_FK_Or	Customers_PrimaryKey	row	CustomerID

Create XML Schema from DB Structure (Oracle)

To follow the example below you will need to have the Oracle Client installed on your PC or network. Please ensure that your Oracle client installation also includes '**Oracle Windows Interfaces'**. Please contact your database administrator on how to install the Oracle Client.

To create a hierarchical schema from a database file (Oracle):

1. Select the menu option **Convert | Create XML Schema from DB Structure.**
2. Select **Oracle (OCI)**, and click **Next**.
   This opens the Oracle Login dialog box.
3. Enter the login parameters and click **OK**.



Create Schema (Oracle Native Connection, Oracle)

Database Owners:

	Owner
□	CTXSYS
□	SYS
☑	WKSYS
□	OLAPSYS
□	MDSYS

Database Tables:

	Table □ Views □ System
☑	WK$CHARSET
☑	WK$CRAWLER_CONFIG_DEFAULT
☑	WK$LANG
☑	WK$MIMETYPES
☑	WK$SYS_CONFIG

Selected Tables:

Owner	Table
WKSYS	WK$CHARSET
WKSYS	WK$CRAWLER_CONFIG_DEFAULT
WKSYS	WK$LANG

Create Schema Options

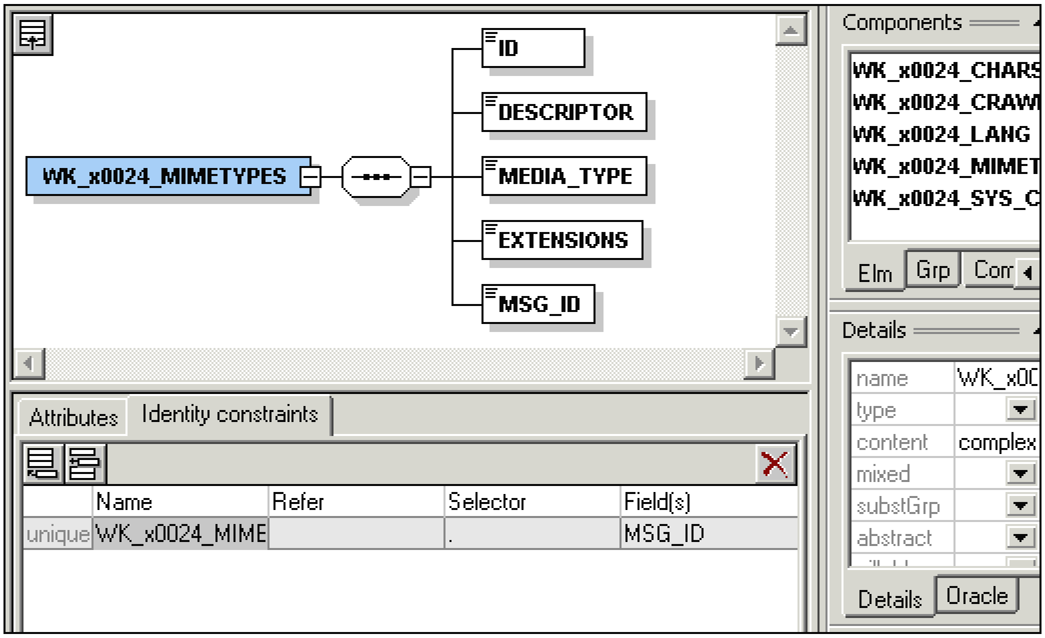
Select a format for resulting schema:
○ Flat (SQL/XML Standard)
● Hierarchical

Import Columns as Elements/Attributes
● Elements by default
○ Attributes by default

Select the schema extension type :
○ None
○ SQL XML
○ MS SQL Server
● Oracle

Select the constraints to generate:
☑ Primary Keys
☑ Foreign Keys
☑ Unique Keys

4. Click the **Hierarchical** radio button.
5. Click the **Owner** entry in the Database Owner list box (**WKSYS** in this example), and select the tables you want to appear in the schema.
6. Click the **Create Schema** button to start the conversion process.



	Name	Refer	Selector	Field(s)
unique	WK_x0024_MIME		.	MSG_ID

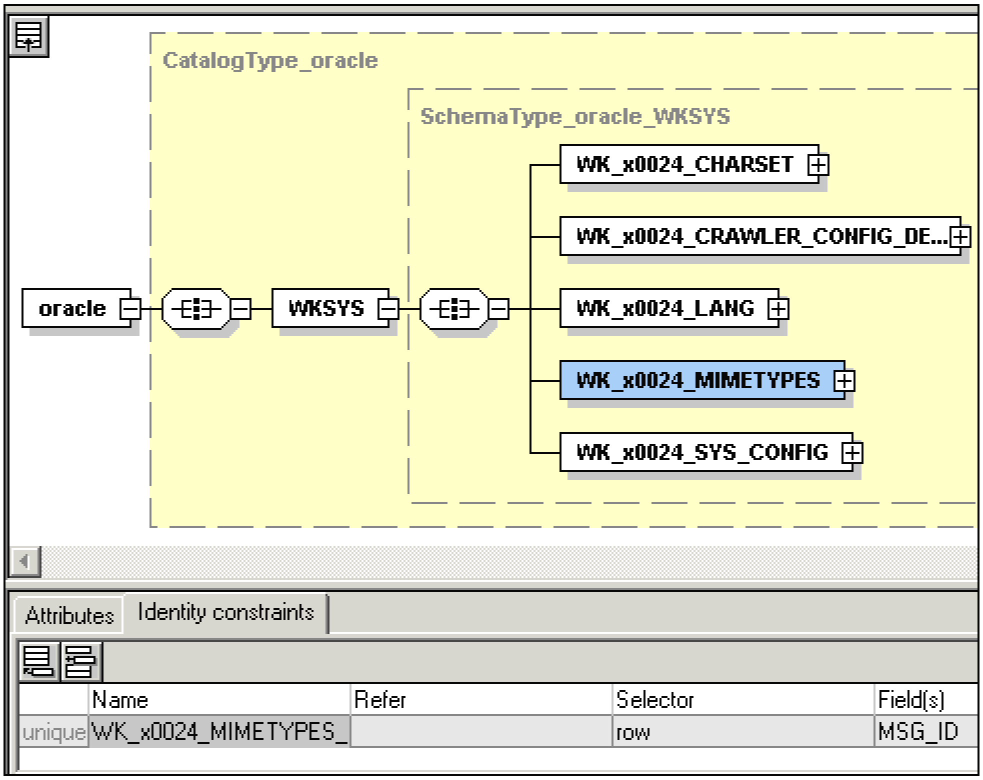
name	WK_x0C
type	
content	complex
mixed	
substGrp	
abstract	

7. Click a component icon to see the content model of that global component. Click the **Identity constraints** tab, to see the keyref and key fields of the respective elements.

   Please note: The Oracle Server Extensions are available in the **Oracle** tab in the Details entry helper.

Result when generating a flat schema model:

If you select the **Flat (SQL/XML Standard)** option in the Create Schema dialog box, the resulting schema appears as shown below.



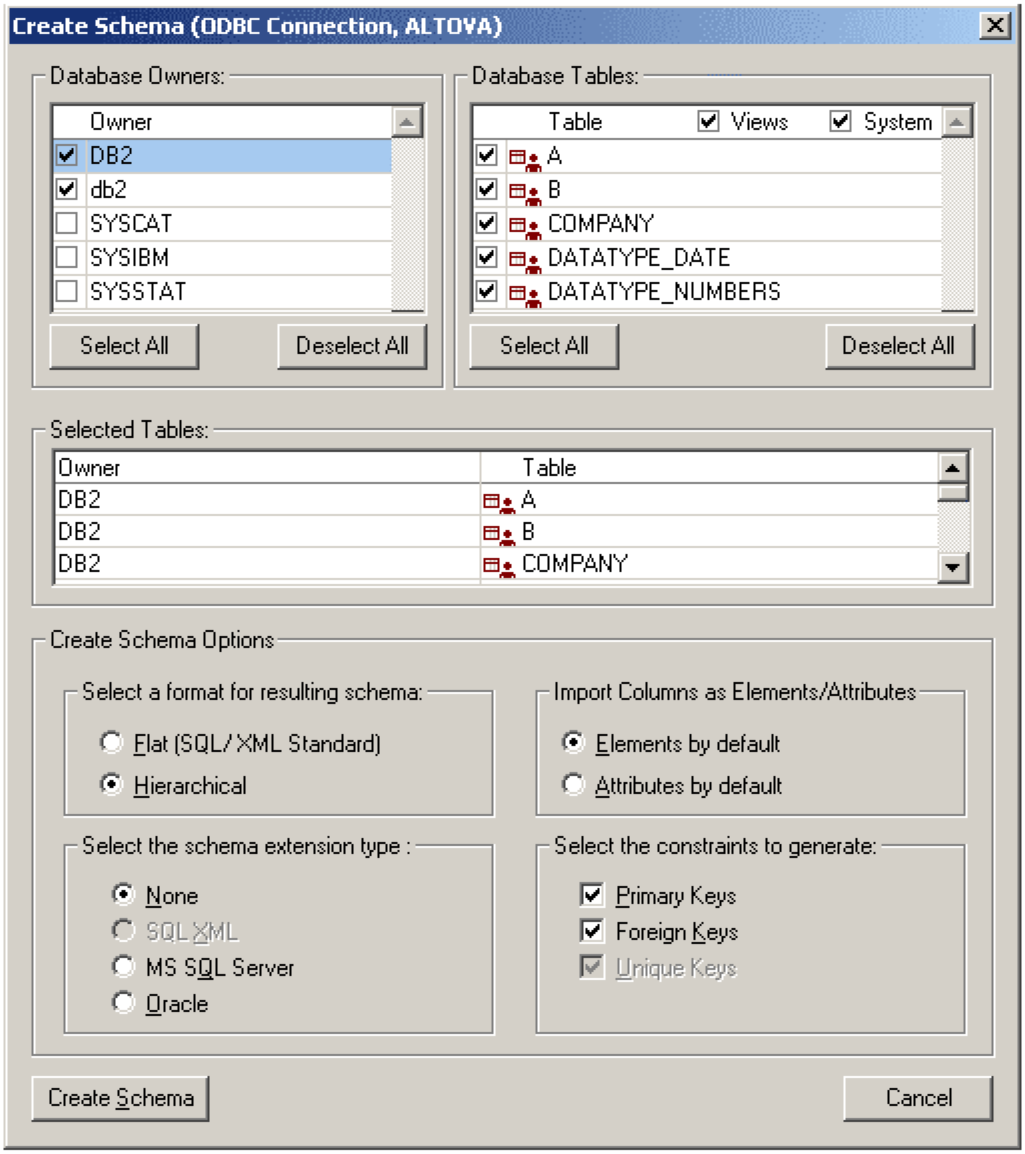
	Name	Refer	Selector	Field(s)
unique	WK_x0024_MIMETYPES_		row	MSG_ID

Create XML Schema from DB Structure (IBM DB2)

To follow the example below you will need to have the IBM DB2 Client installed on your PC or network.

To create a hierarchical schema from a database file (IBM DB2):

1. Select the menu option **Convert | Create XML Schema from DB Structure.**
2. Select **IBM DB2 (ODBC)**, and click **Next**.
   The **IBM DB2 : ODBC Connection String** dialog box is opened.
3. Click the **Build** button to select the database connection settings.
   This opens the Select Data Source dialog box.
4. Click the **Machine Data Source** tab, select the data source name created when you installed the IBM DB2 client (Altova), and click **OK**.
   The Connect to DB2 dialog box is opened.
5. Enter the **User ID** and **password**, and click **OK** to connect to the database.
   The connection string is created and placed in the connection string text box.
6. Click **Next** to continue.
   This opens the **Import Database Data** dialog box.



Create Schema (ODBC Connection, ALTOVA)

Database Owners:

	Owner
☑	DB2
☑	db2
□	SYSCAT
□	SYSIBM
□	SYSSTAT

Database Tables:

	Table ☑ Views ☑ System
☑	A
☑	B
☑	COMPANY
☑	DATATYPE_DATE
☑	DATATYPE_NUMBERS

Selected Tables:

Owner	Table
DB2	A
DB2	B
DB2	COMPANY

Create Schema Options

Select a format for resulting schema:

○ Flat (SQL/XML Standard)
● Hierarchical

Import Columns as Elements/Attributes

● Elements by default
○ Attributes by default

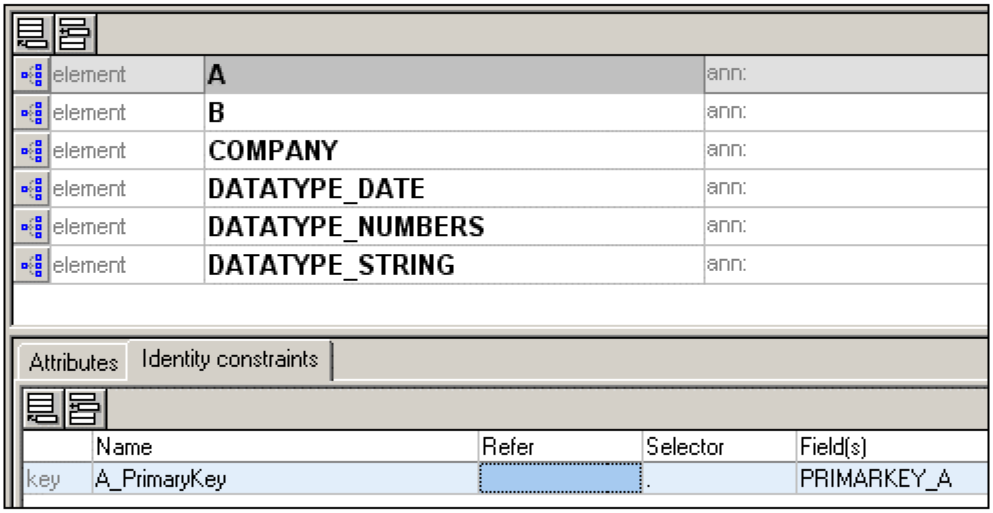
Select the schema extension type :

● None
○ SQL XML
○ MS SQL Server
○ Oracle

Select the constraints to generate:

☑ Primary Keys
☑ Foreign Keys
☑ Unique Keys

7. Click the **Owner** entry in the Database Owner list box and select the tables you want to appear in the schema.
8. Click the **Create Schema** button to start the conversion process.
9. Click a component icon to see the content model of that global component.

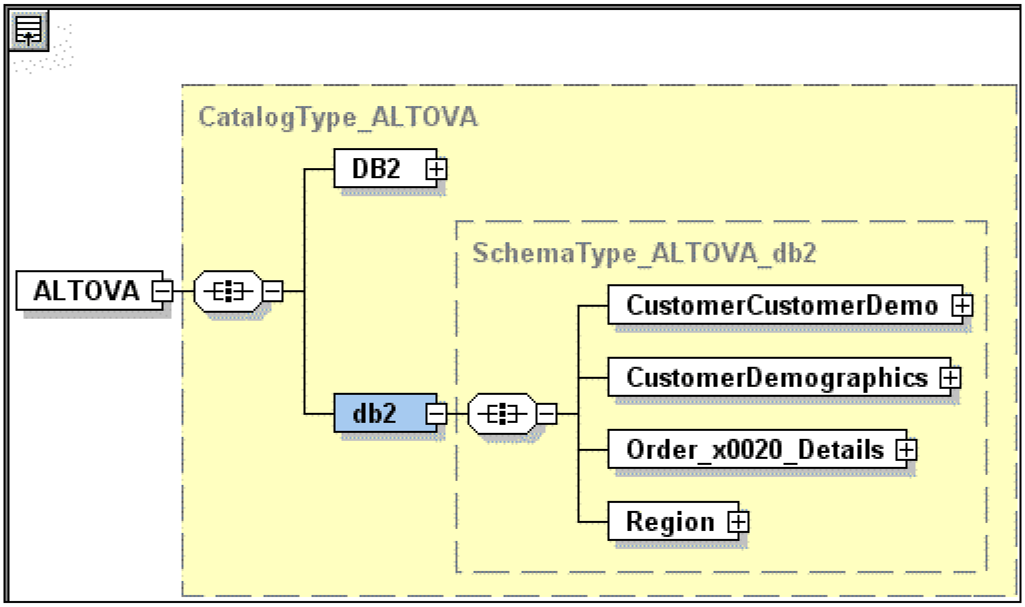


element	A	ann:
element	B	ann:
element	COMPANY	ann:
element	DATATYPE_DATE	ann:
element	DATATYPE_NUMBERS	ann:
element	DATATYPE_STRING	ann:

	Name	Refer	Selector	Field(s)
key	A_PrimaryKey		.	PRIMARKEY_A

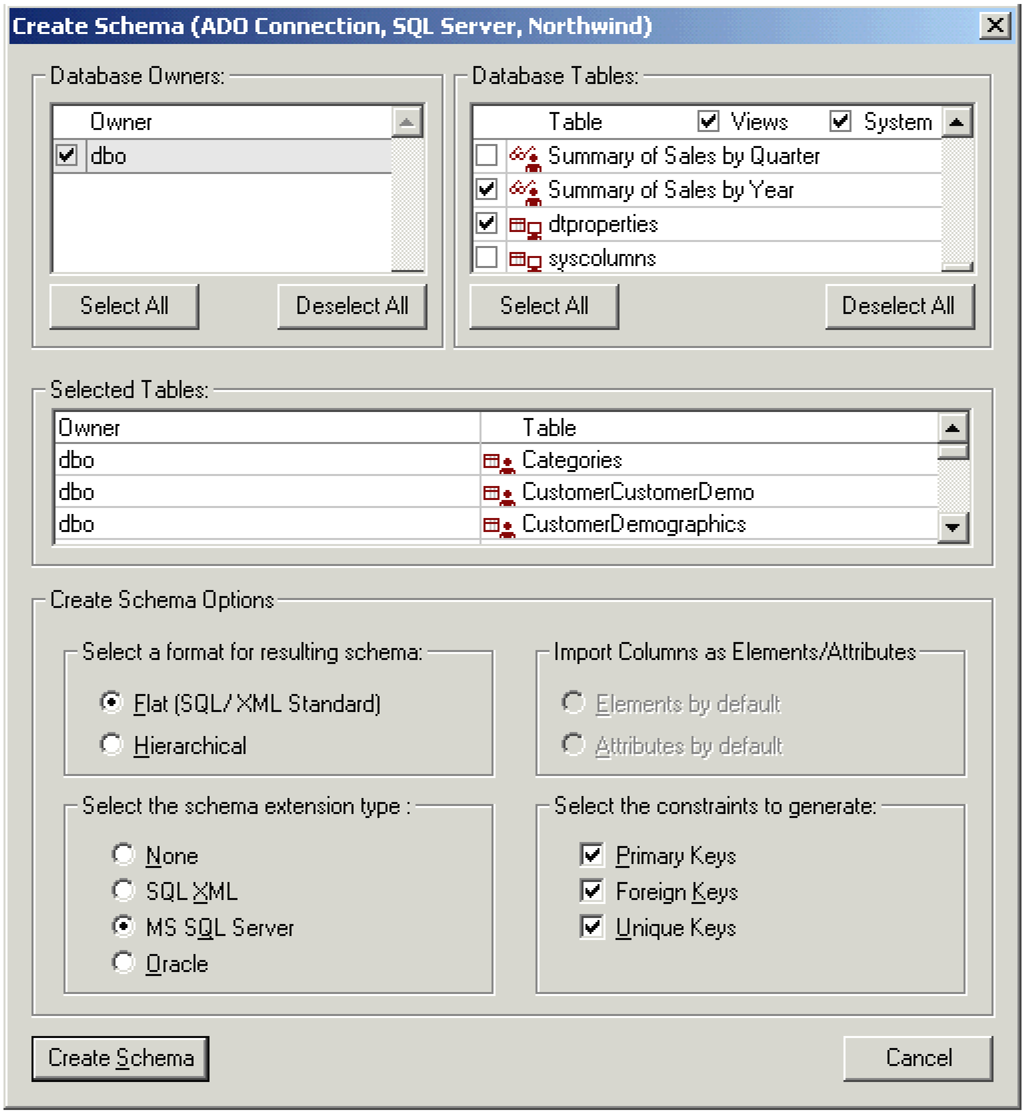
Result when generating a flat schema model:

If you select the **Flat (SQL/XML Standard)** option in the Create Schema dialog box, the resulting schema appears as shown below.



Create XML Schema options

XML Schema formats:



Create Schema (ADO Connection, SQL Server, Northwind)

Database Owners:

	Owner
☑	dbo

Database Tables:

	Table ☑ Views ☑ System
□	Summary of Sales by Quarter
☑	Summary of Sales by Year
☑	dtproperties
□	syscolumns

Selected Tables:

Owner	Table
dbo	Categories
dbo	CustomerCustomerDemo
dbo	CustomerDemographics

Create Schema Options

Select a format for resulting schema:

● Flat (SQL/XML Standard)
○ Hierarchical

Import Columns as Elements/Attributes

○ Elements by default
○ Attributes by default

Select the schema extension type :

○ None
○ SQL XML
● MS SQL Server
○ Oracle

Select the constraints to generate:

☑ Primary Keys
☑ Foreign Keys
☑ Unique Keys

- 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.

- 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

Null constraints and schema Default Values
If the column has the constraint NULL, then the attribute "Nillable=true" is appended to the elements and "Use=optional" for attributes. Default values are generated only for columns that are a string or number type.

There is an additional restriction when generating the columns as attributes with the Hierarchical format. If there is a default value defined and the "User" attribute is "required", the default value is not generated.

Tables and Views
There are four resulting table types that can be retrieved from a database. These are:

- User Tables
- System Tables
- User Views
- System Views

The different table types are identified by icons that appear next to the selected tables and owner names (if the database provides them).

Please note:
When connecting to a database via ODBC, the distinction between user and system (only for views) cannot be maintained. All tables are therefore user tables, and all views are user views.

MS Access System Tables are not displayed, because it is currently not possible to retrieve these tables.

Owners
Not all databases support the concept of owners. For example MS Access does not provide any owners for its databases. In this case only the tables found in the database are listed and generated, and thus "Flat schema" format is not supported.

Identity Constraints and Relationships
Identity constraints are retrieved and relationships are generated via ADO, ODBC and OCI Importing methods.

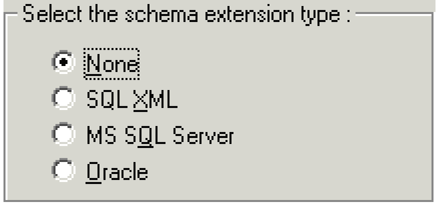
When importing a MS Access database via ODBC, no identity constraints are retrieved because MS Access does not support this functionality via ODBC.

When working with foreign keys in MySQL, it is important to note that the tables that are created, must be of the type INNODB (the specific MySQL version must also support this type). This is an inherent limitation of the MySQL database. For more information please see: http://www.mysql.com/doc/en/InnoDB_foreign_key_constraints.html

Unique keys are currently not retrieved/supported via ODBC.

Generating schema extensions

Schema Extensions



Select the schema extension type :

● None
○ SQL XML
○ MS SQL Server
○ Oracle

Schema extension information is additional information read from a database that is then embedded in the schema as either:

- annotation data, or
- as attributes.

There are four choices when generating schemas:

- No Extensions information
- SQL Server Extensions
- Oracle Extensions
- SQL/XML Extensions

None:
Selecting **None** causes no additional information to be provided by the database.

SQL Server Extensions:
Selecting Microsoft SQL Server, generates SQL Server extensions. Please see "Using Annotations in XSL Schemas" in the the SQL Server Books Online (Updated –SP3), 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 are stored as attributes. A full description of these attributes can be found here:
http://download-west.oracle.com/docs/cd/B10501_01/appdev.920/a96620/xdb05obj.htm#1027227 (You need to have access to the Oracle Technology network, to access this page.)

The following subset of attributes is currently generated:

- SQLName
- SQLType
- SQLSchema

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.

Although the Oracle and SQL Server 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 Oracle or SQL Server.

Datatype conversions

See Appendix.

11.9.5 DB Import based on XML Schema

XMLSpy allows you to create an XML document containing database data based on an existing XML Schema. You must currently create the schema (on which the import is based) using the menu option **Convert | Create XML Schema from DB Structure.**

The following databases are supported:

- Microsoft Access 2000 and 2003
- Microsoft SQL Server
- Oracle
- MySQL
- Sybase
- IBM DB2

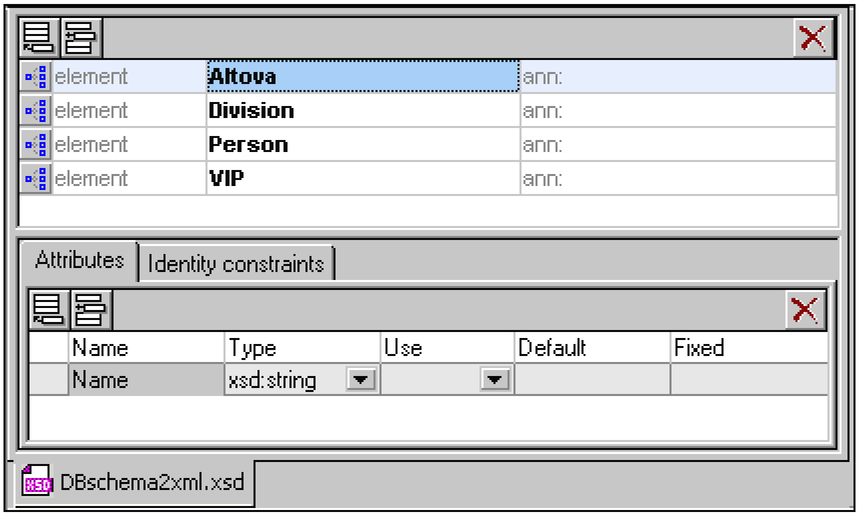
The source database can be a Microsoft Access (2000 or 2003) database, or you can build an ADO connection string to connect to the database of your choice. Note that only ADO connections can be created to any of the databases using this option. When working with Oracle, MySQL, Sybase and IBM DB2, it is recommended to use the MS OLE-DB Provider for ODBC Drivers to connect to the respective database.

Specifying the data you want to import

You can specify the data set you want to import by defining which table is to act as the root table of the imported data set. This then **automatically** creates a shape command which is placed in the Selection statement text box. The database data is then imported with all hierarchical relationships intact.

To import XML data from a Microsoft Access database:

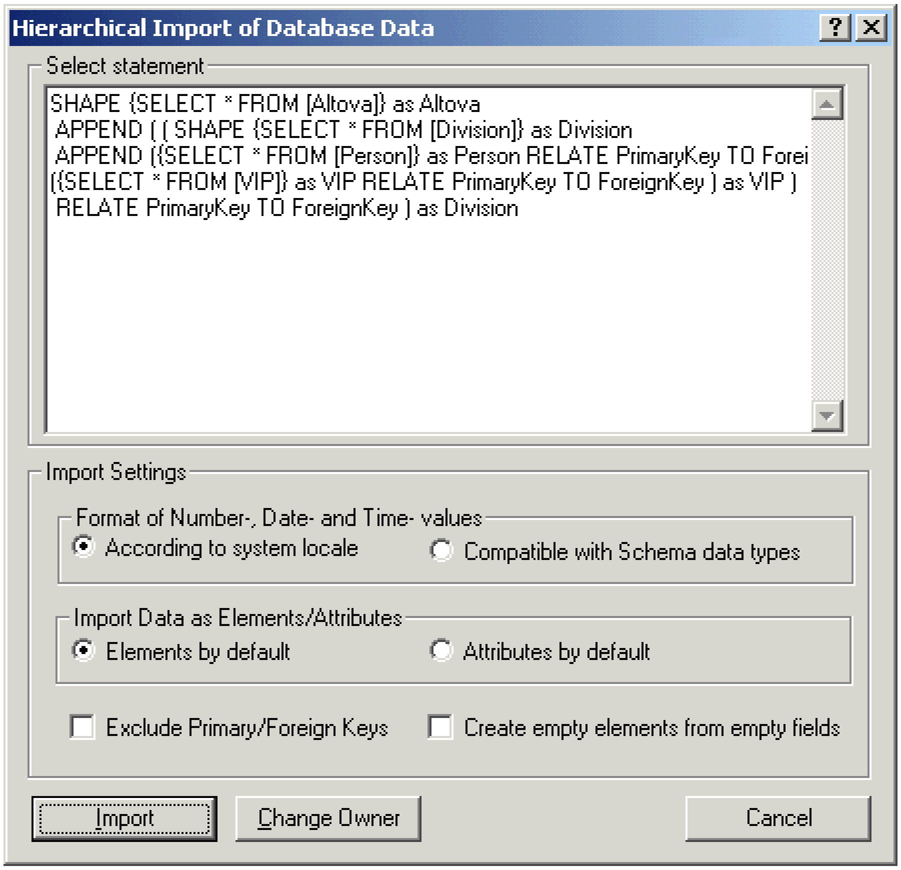
1. Open the **DBschema2xml.xsd** schema file supplied with XMLSpy (`Tutorial` folder).



element	**Altova**	ann:
element	**Division**	ann:
element	**Person**	ann:
element	**VIP**	ann:

Name	Type	Use	Default	Fixed
Name	xsd:string			

2. Select the menu option **Convert | DB Import based on XML Schema**.
   This opens the Select a Source Database dialog box.
   Select the **Microsoft Access (ADO)** option and click **Next**.
   Click **Browse** to select an existing MS Access database (**DB2schema.mdb**), then click **Next**.
3. Select a root table from dialog box, e.g., **Altova**.
   This is where you define which tables you want to import (Altova in this case). The root table becomes the top level table in the table hierarchy you import.
4. Click **OK** once you have selected the root table.



```
SHAPE {SELECT * FROM [Altova]} as Altova
 APPEND ( ( SHAPE {SELECT * FROM [Division]} as Division
 APPEND ({SELECT * FROM [Person]} as Person RELATE PrimaryKey TO Forei
({SELECT * FROM [VIP]} as VIP RELATE PrimaryKey TO ForeignKey ) as VIP )
 RELATE PrimaryKey TO ForeignKey ) as Division
```

Import Settings

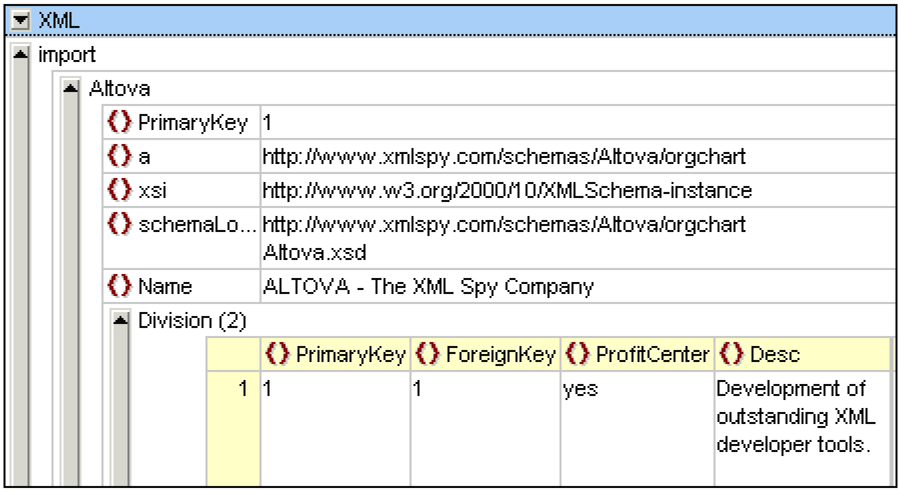
Format of Number-, Date- and Time- values
● According to system locale ○ Compatible with Schema data types

Import Data as Elements/Attributes
● Elements by default ○ Attributes by default

□ Exclude Primary/Foreign Keys □ Create empty elements from empty fields

The select statement, based on your root element choice, is automatically generated and placed in the Select statement text box. You can edit the select statement, to change the import parameters if you wish.

5. Click the **Import** button to import the database data.
   The XML document appears as an "Untitledx.xml" file in the main window.



XML

import

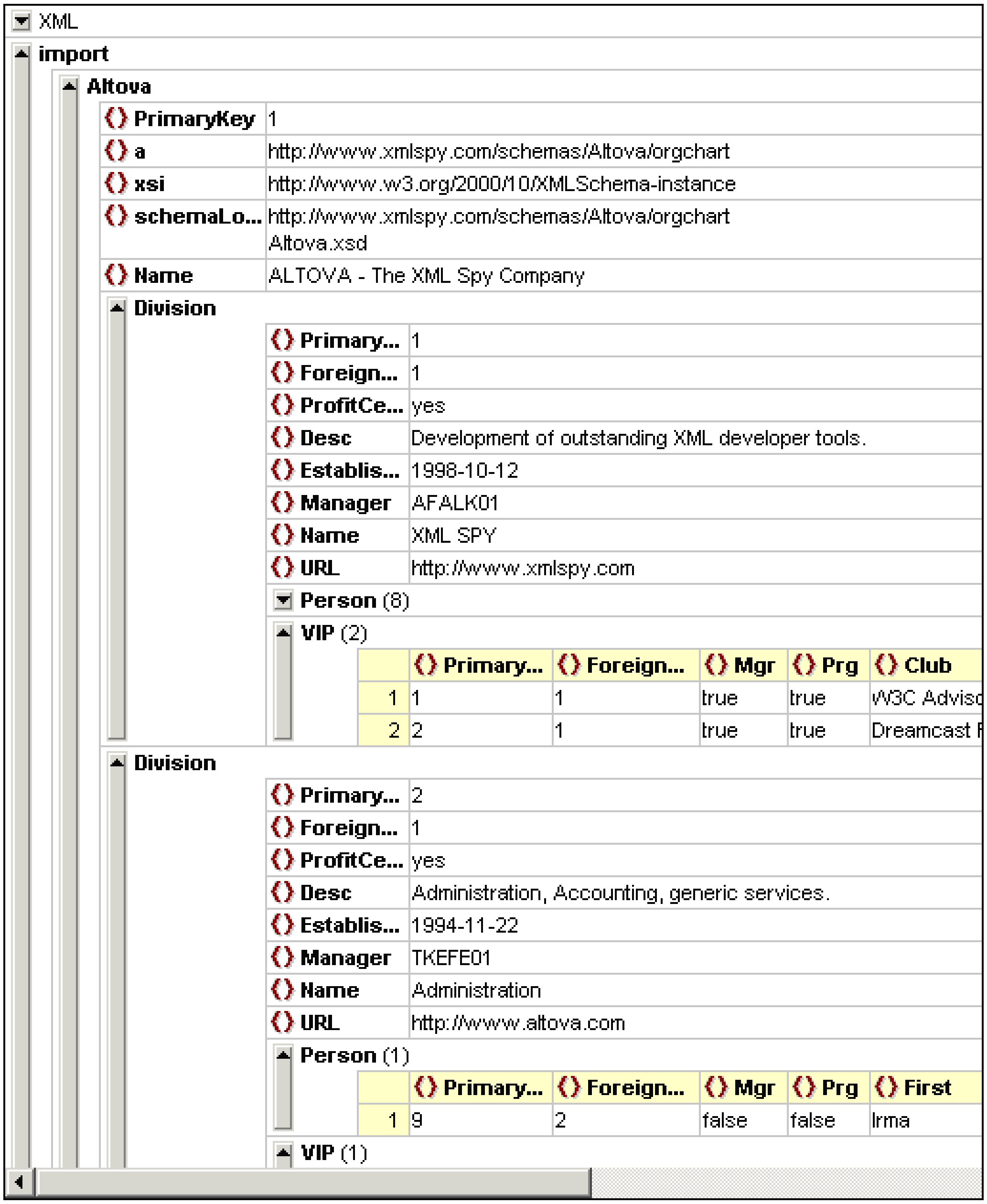
Altova

PrimaryKey	1
a	http://www.xmlspy.com/schemas/Altova/orgchart
xsi	http://www.w3.org/2000/10/XMLSchema-instance
schemaLo...	http://www.xmlspy.com/schemas/Altova/orgchart Altova.xsd
Name	ALTOVA - The XML Spy Company

Division (2)

	PrimaryKey	ForeignKey	ProfitCenter	Desc
1	1	1	yes	Development of outstanding XML developer tools.

6. Click the Altova element and select the menu option **XML | Table | Display as table** to remove the table formatting and display the data vertically. The Division elements were treated the same way to produce the screen shot below.



XML

import

Altova

PrimaryKey	1
a	http://www.xmlspy.com/schemas/Altova/orgchart
xsi	http://www.w3.org/2000/10/XMLSchema-instance
schemaLo...	http://www.xmlspy.com/schemas/Altova/orgchart Altova.xsd
Name	ALTOVA - The XML Spy Company

Division

Primary...	1
Foreign...	1
ProfitCe...	yes
Desc	Development of outstanding XML developer tools.
Establis...	1998-10-12
Manager	AFALK01
Name	XML SPY
URL	http://www.xmlspy.com

Person (8)

VIP (2)

	Primary...	Foreign...	Mgr	Prg	Club
1	1	1	true	true	W3C Adviso
2	2	1	true	true	Dreamcast F

Division

Primary...	2
Foreign...	1
ProfitCe...	yes
Desc	Administration, Accounting, generic services.
Establis...	1994-11-22
Manager	TKEFE01
Name	Administration
URL	http://www.altova.com

Person (1)

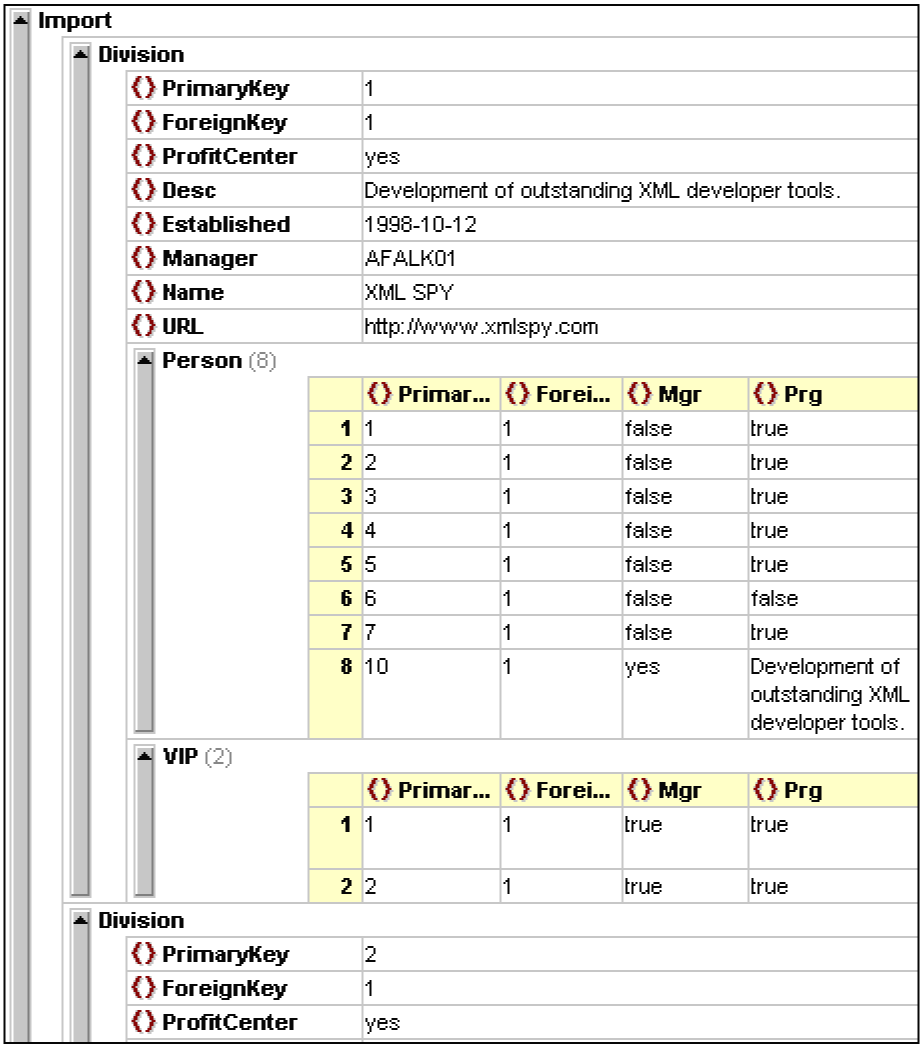
	Primary...	Foreign...	Mgr	Prg	First
1	9	2	false	false	Irma

VIP (1)

In this case the complete database was imported, the "root" table being Altova. The item Altova Name, was imported as an attribute in keeping with the schema definition `DBschema2xml.xsd` (visible in the attribute tab of the schema at the beginning of this section).

Importing a partial data set:

1. Repeat the above sequence and select **Division** as the root table.
2. Use the **DB2schema.mdb** file as the data source.
   The screenshot shows the resulting XML document. The Person table is expanded and the Division table formatting is disabled for a better overview.



Import

Division

PrimaryKey	1
ForeignKey	1
ProfitCenter	yes
Desc	Development of outstanding XML developer tools.
Established	1998-10-12
Manager	AFALK01
Name	XML SPY
URL	http://www.xmlspy.com

Person (8)

	Primar...	Forei...	Mgr	Prg
1	1	1	false	true
2	2	1	false	true
3	3	1	false	true
4	4	1	false	true
5	5	1	false	true
6	6	1	false	false
7	7	1	false	true
8	10	1	yes	Development of outstanding XML developer tools.

VIP (2)

	Primar...	Forei...	Mgr	Prg
1	1	1	true	true
2	2	1	true	true

Division

PrimaryKey	2
ForeignKey	1
ProfitCenter	yes

MySQL connection issue:

When creating the ADO connection based on ODBC, please do not use a File DSN. File DSN causes the error message "Class not registered"; instead please use either User or System DSN.

11.9.6 Create DB Structure from XML Schema

XMLSpy allows you to create an empty database (or skeleton database) based on an existing schema file. The 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 XML Schema from DB Structure** menu option.

	Create DB Structure from XML Schema		
	Directly	using ODBC	using ADO
Microsoft 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.

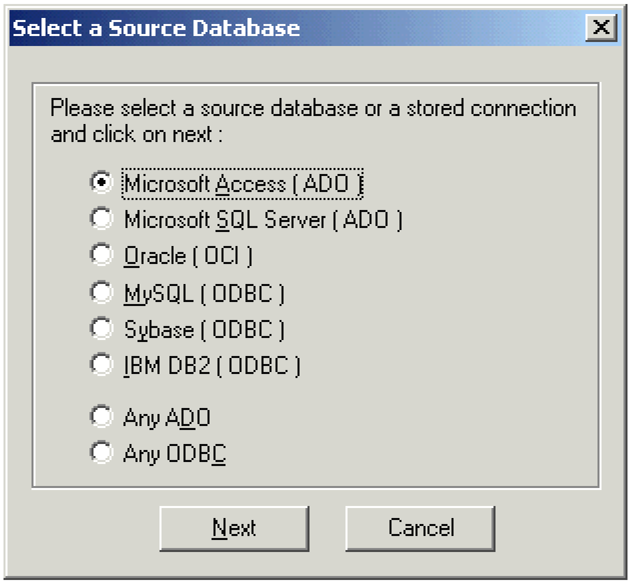
- SQL Server DB, Oracle DB, or Sybase DB:
  A schema that was generated in flat format, for one of the above databases, will have the schema catalog name extracted and used in the generated SQL script as the DB name. This means that the resulting SQL script will be executed on a target DB whose name 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.

The method described below, is generally the same for each type of database. Please also see the section Mapping from XML Schema datatypes for specific information on how database specific datatypes are mapped to individual schema datatypes.

To create a database based on a schema:

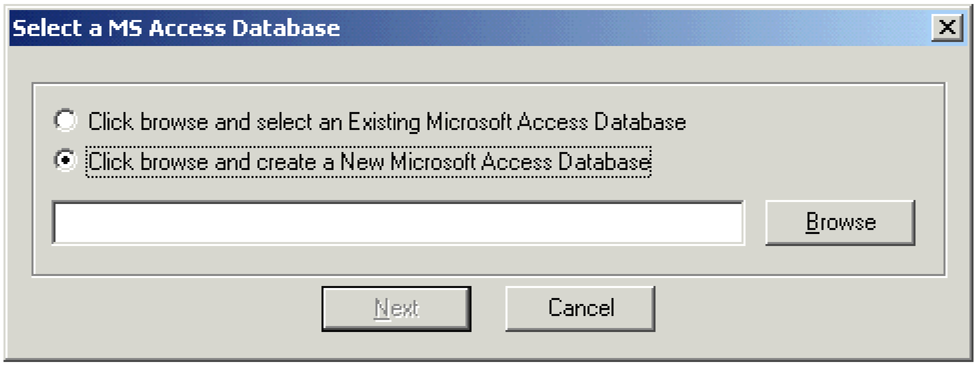
1. Open the schema file **No-constraints.xsd** in the `Tutorial` folder.
2. Make sure you are in Schema/WSDL view
3. Select the menu option **Convert | Create DB Structure from XML Schema**.
4. Click the **Microsoft Access** radio button, and then click **Next**.



Please select a source database or a stored connection and click on next :

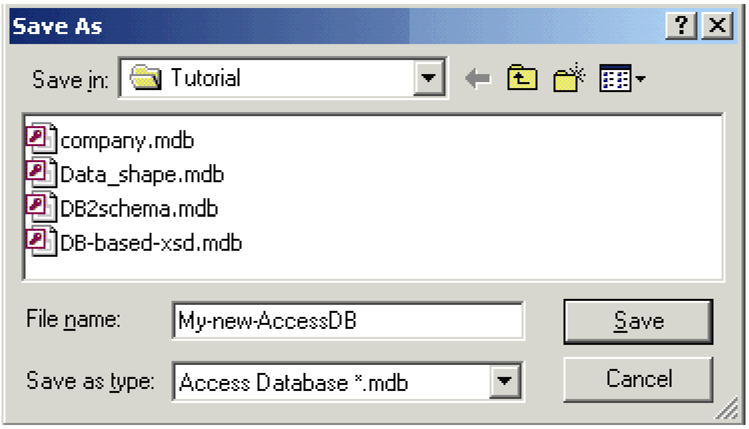
● Microsoft Access (ADO)
○ Microsoft SQL Server (ADO)
○ Oracle (OCI)
○ MySQL (ODBC)
○ Sybase (ODBC)
○ IBM DB2 (ODBC)
○ Any ADO
○ Any ODBC

This opens the **Select an MS Access Database** dialog box. You can now decide if you want to overwrite an existing database, or create a new one.
Note that XMLSpy supports MS Access 2000 and MS Access 2003 databases.



○ Click browse and select an Existing Microsoft Access Database
● Click browse and create a New Microsoft Access Database

5. Click the **Browse and create a New ...** radio button, then click **Browse**.



Save in: Tutorial

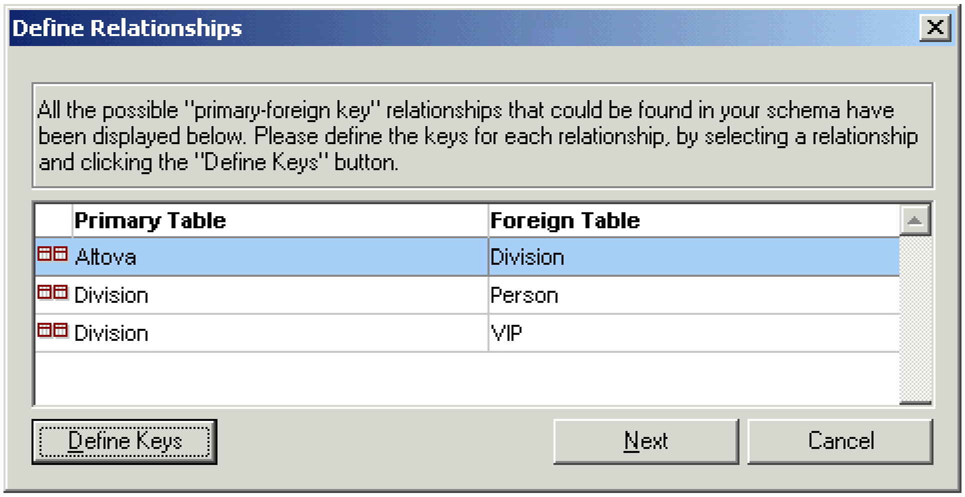
company.mdb
Data_shape.mdb
DB2schema.mdb
DB-based-xsd.mdb

File name: My-new-AccessDB
Save as type: Access Database *.mdb

6. Enter the name of the Database you want to create (e.g., My-new AccessDB) and click **Save**.
This returns you to the previous dialog box, with the database name and path entered in

the text field.

7. Click **Next**.
   The **Define Relationships** dialog box opens if the schema you are using as a basis does **not** have any predefined Identity constraints.



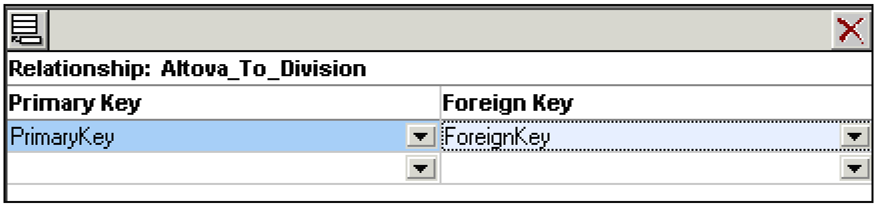
Define Relationships

All the possible "primary-foreign key" relationships that could be found in your schema have been displayed below. Please define the keys for each relationship, by selecting a relationship and clicking the "Define Keys" button.

Primary Table	Foreign Table
Altova	Division
Division	Person
Division	VIP

The file used in this example does not have predefined ID constraints, so the relationships have to be defined manually.
Please note: This can be achieved by selecting **Convert | Create XML Schema from database** and **deselecting** all three **Identity constraint** check boxes.)

8. Select the first row and click the **Define Keys** button (or double-click a row).



Relationship: Altova_To_Division

Primary Key	Foreign Key
PrimaryKey	ForeignKey

The first column allows you to define the primary key field of the Division table

9. Click the combo box and select the primary key field, in this case "PrimaryKey".
   The second column allows you to define the foreign key field of the Division table.
10. Click the combo box and select the foreign key field, in this case "ForeignKey".

    Please note:
    The **Define Keys** button changes to **List Relationships**, when defining a specific relationship. Clicking **List Relationships** takes you back to the relationship list.

11. Define the relationships for the rest of the tables and click **Next**. It is not, however, mandatory to define any, or all, of the relationships.

    The **Preview Database** dialog box is opened. The **Preview Database Structure** tab gives you a global overview of the current database structure, while the **Edit SQL tab** displays the actual SQL commands used to create all database tables.

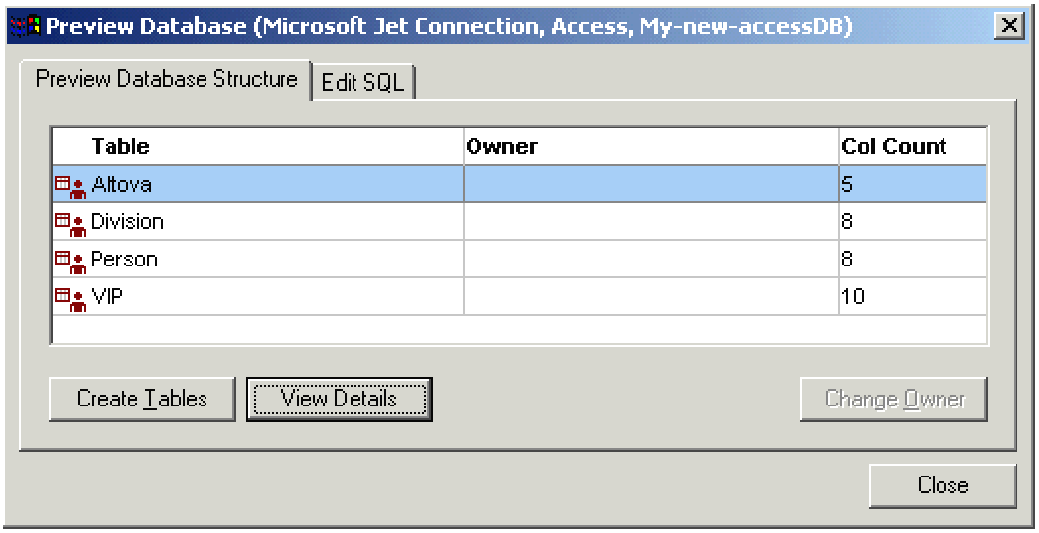

Preview Database (Microsoft Jet Connection, Access, My-new-accessDB)
Preview Database Structure
Edit SQL
Create Tables
View Details
Change Owner
Close

Table	Owner	Col Count
Altova		5
Division		8
Person		8
VIP		10

12. Click the **View Details** button to see the Altova table field definitions.

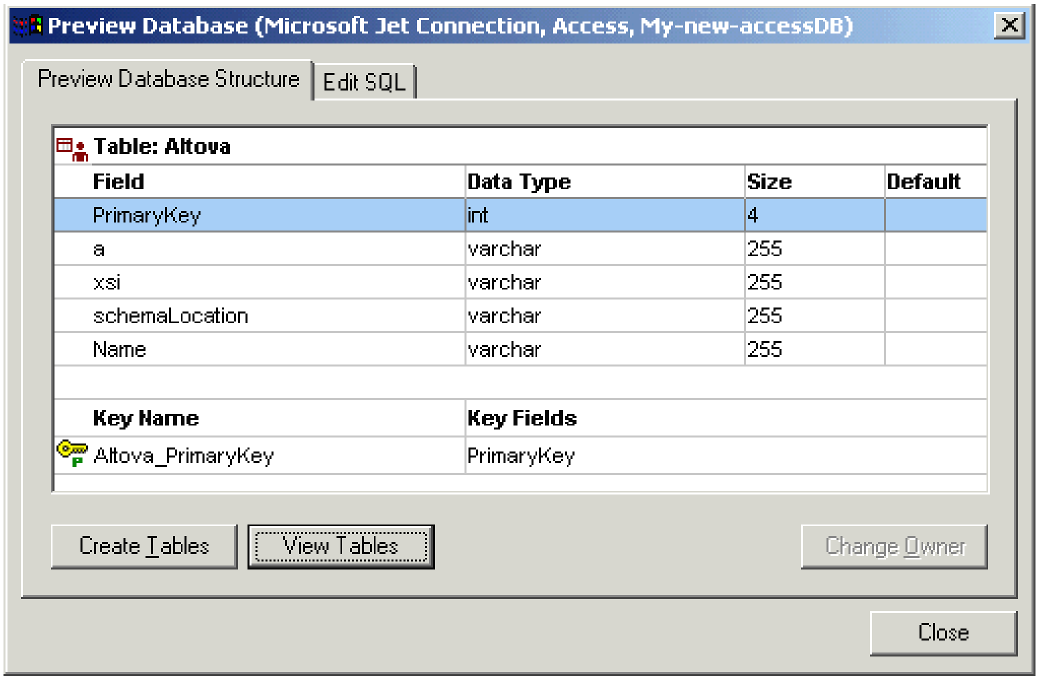
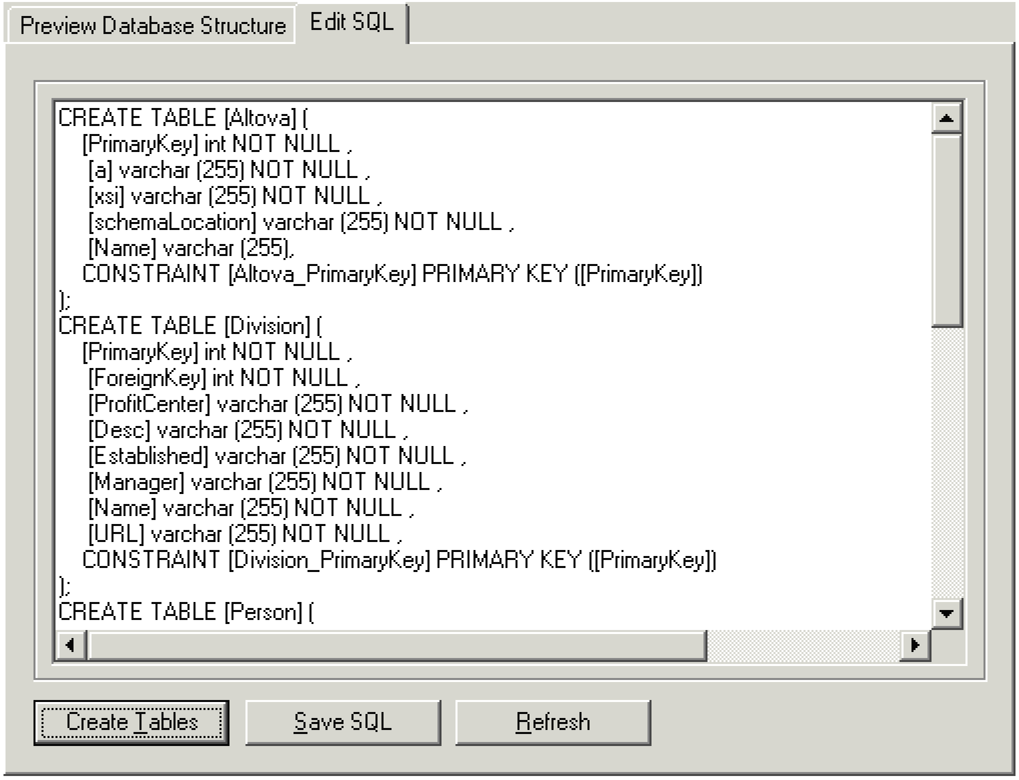

Preview Database (Microsoft Jet Connection, Access, My-new-accessDB)
Preview Database Structure
Edit SQL
Create Tables
View Tables
Change Owner
Close

Table: Altova

Field	Data Type	Size	Default
PrimaryKey	int	4	
a	varchar	255	
xsi	varchar	255	
schemaLocation	varchar	255	
Name	varchar	255	

Key Name	Key Fields
Altova_PrimaryKey	PrimaryKey

13. Click the **View Tables** button to go back to the table overview.
14. Click the **Edit SQL** tab to see and/or edit the current SQL statements.



```
CREATE TABLE [Altova] (
   [PrimaryKey] int NOT NULL ,
    [a] varchar (255) NOT NULL ,
    [xsi] varchar (255) NOT NULL ,
    [schemaLocation] varchar (255) NOT NULL ,
    [Name] varchar (255),
   CONSTRAINT [Altova_PrimaryKey] PRIMARY KEY ([PrimaryKey])
);
CREATE TABLE [Division] (
   [PrimaryKey] int NOT NULL ,
    [ForeignKey] int NOT NULL ,
    [ProfitCenter] varchar (255) NOT NULL ,
    [Desc] varchar (255) NOT NULL ,
    [Established] varchar (255) NOT NULL ,
    [Manager] varchar (255) NOT NULL ,
    [Name] varchar (255) NOT NULL ,
    [URL] varchar (255) NOT NULL ,
   CONSTRAINT [Division_PrimaryKey] PRIMARY KEY ([PrimaryKey])
);
CREATE TABLE [Person] (
```

The **Edit SQL** tab allows you to **edit** the automatically generated SQL select statement visible here. The **Save SQL** button saves the SQL statement to a file, while the **Refresh** button resets any changes made in this tab.

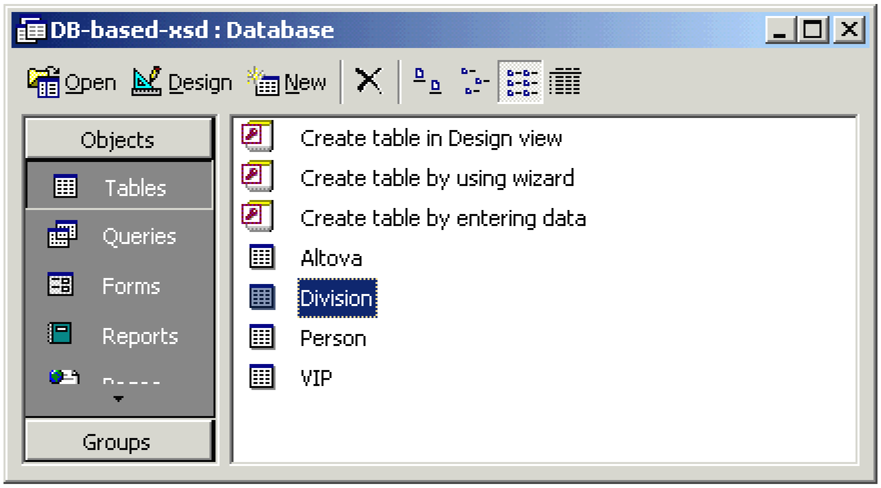
15. Click the **Create Tables** button to generate the Access database with the settings you have defined. A message appears, stating if the generation was successful or not.

Please note:

Changes made to the SQL statement in this tab, are discarded when you click the **Preview Database Structure** tab. If you want to have the modified SQL statement produce the database tables, make sure that you click the **Create Tables** button in this (Edit SQL) tab.

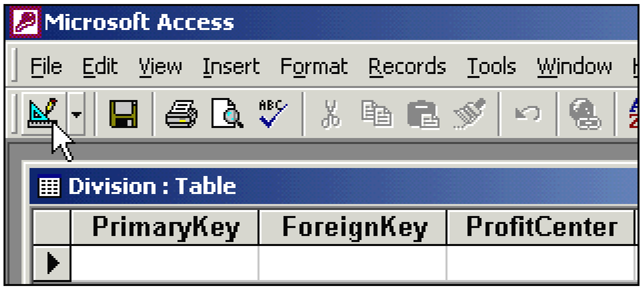
Viewing the generated Database:

1. Double-click the **MY-new-accessDB.mdb** file in Explorer, or open the file in MS Access.



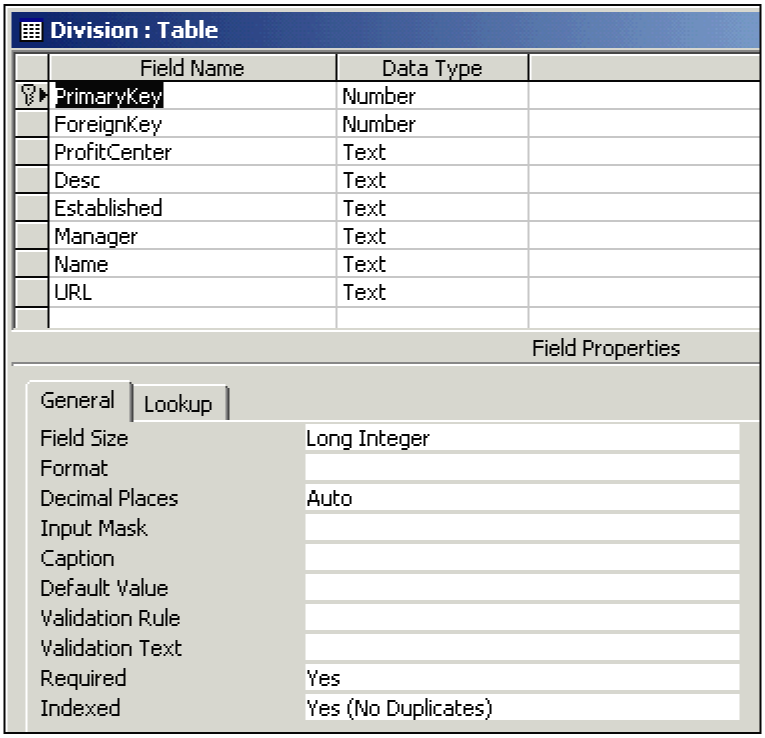
Create table in Design view
Create table by using wizard
Create table by entering data
Altova
Division
Person
VIP

2. Double-click the **Division** entry to see the Division Table.



PrimaryKey	ForeignKey	ProfitCenter

3. Select the menu option **View | Design view**, or click the "View" icon to change into the Design view.



Field Name	Data Type
PrimaryKey	Number
ForeignKey	Number
ProfitCenter	Text
Desc	Text
Established	Text
Manager	Text
Name	Text
URL	Text

Field Size	Long Integer
Format	
Decimal Places	Auto
Input Mask	
Caption	
Default Value	
Validation Rule	
Validation Text	
Required	Yes
Indexed	Yes (No Duplicates)

The Design view supplies information on the field properties of the database table: the field names, data types, etc. The parameter settings you see here have been transferred from the schema definition.

Datatype conversions

See Appendix.

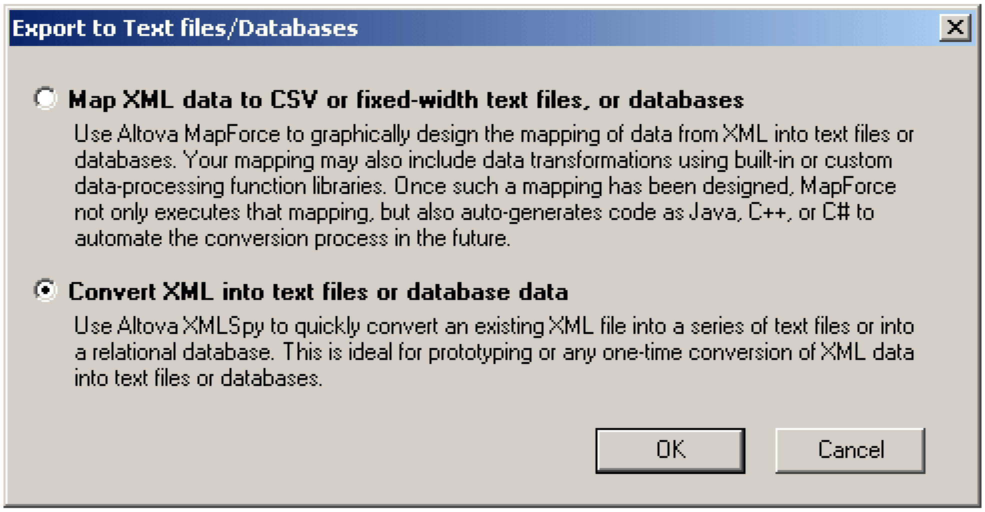
11.9.7 Export to Text files / Database...

The command **Convert | Export to Text files / Database** exports XML data into other formats for exchange with databases or legacy systems.

Depending on the output data format required, you may want to use either XSLT Transformations or this Export command to export your XML data.

You first need to define the structure of the data to be exported. Since XML is structured hierarchically and most database and legacy systems use the relational model, XMLSpy will help you in producing output that can be interpreted in a relational context.

When you select this command, the following dialog appears:



Export to Text files/Databases

○ **Map XML data to CSV or fixed-width text files, or databases**

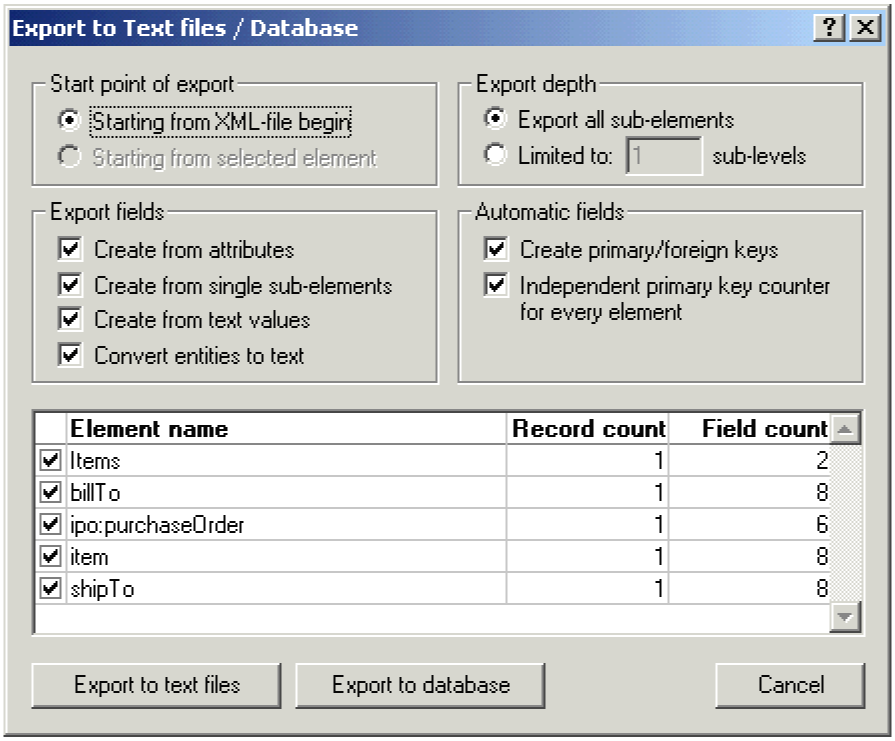
Use Altova MapForce to graphically design the mapping of data from XML into text files or databases. Your mapping may also include data transformations using built-in or custom data-processing function libraries. Once such a mapping has been designed, MapForce not only executes that mapping, but also auto-generates code as Java, C++, or C# to automate the conversion process in the future.

● **Convert XML into text files or database data**

Use Altova XMLSpy to quickly convert an existing XML file into a series of text files or into a relational database. This is ideal for prototyping or any one-time conversion of XML data into text files or databases.

Select **Convert XML into text files or database data** and click **OK**. The **Export to Text files / Database** dialog appears (*see screenshot*). The settings you can make in this dialog are described below.

Please note: If you select **Map XML data to CSV or fixed-width text files, or databases**, you must have Altova MapForce installed on your computer.



Start point of export
● Starting from XML-file begin
○ Starting from selected element

Export depth
● Export all sub-elements
○ Limited to: 1 sub-levels

Export fields
☑ Create from attributes
☑ Create from single sub-elements
☑ Create from text values
☑ Convert entities to text

Automatic fields
☑ Create primary/foreign keys
☑ Independent primary key counter for every element

	Element name	Record count	Field count
☑	Items	1	2
☑	billTo	1	8
☑	ipo:purchaseOrder	1	6
☑	item	1	8
☑	shipTo	1	8

Start point of export
You can choose to export the entire XML document or restrict your export to the data starting from the currently selected element (and its child items).

Export fields
Depending on your XML data, you may want to export only elements, attributes, or the textual content of your elements into the fields common to structured text files or databases.

The list of available element types, in the preview window, lets you choose which elements you want to export, and also displays how many records and fields this will produce once the export operation is started.

Export depth
You can choose to export all sub-elements, or limit the number of sub-element levels.

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.

When you are finished defining the scope of your export operation, click one of the two **Export** buttons, to export your data to a set of text files or to a database.

Export to Database

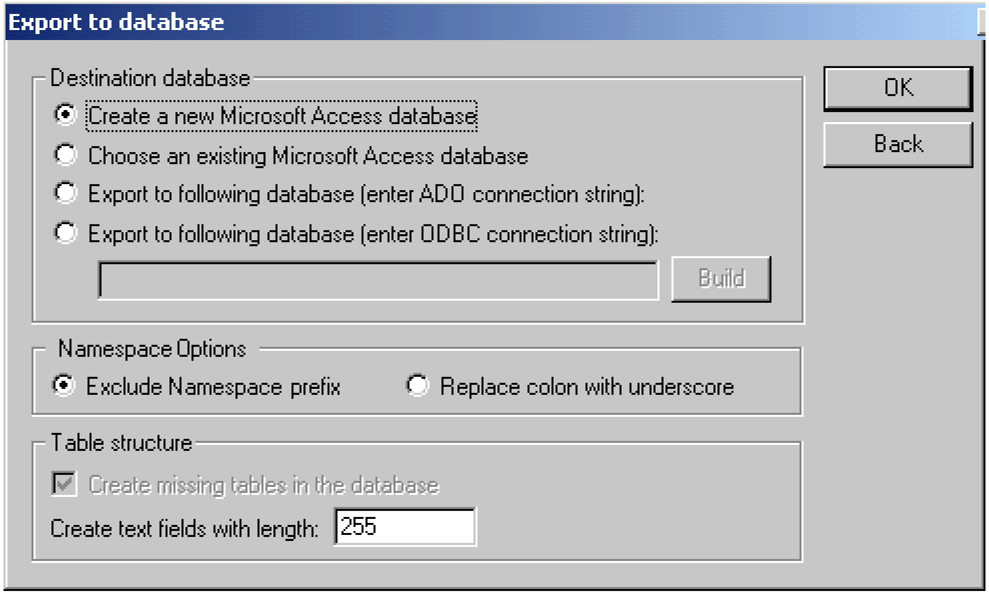
This command button allows you to specify the type of database you want to export your data to.

You can select the destination database and the action to be performed:

- Create a new Microsoft Access (Jet Engine) database file
- Add the data to the tables in an existing Access file, or

- Export the data to any other ODBC or ADO-compliant database system (such as SQL Server or Oracle).

The Namespace options allow you to either exclude the namespace prefix, or replace the namespace prefix colon with an underscore character.



Export to database

Destination database

● Create a new Microsoft Access database

○ Choose an existing Microsoft Access database

○ Export to following database (enter ADO connection string):

○ Export to following database (enter ODBC connection string):

Namespace Options

● Exclude Namespace prefix ○ Replace colon with underscore

Table structure

☑ Create missing tables in the database

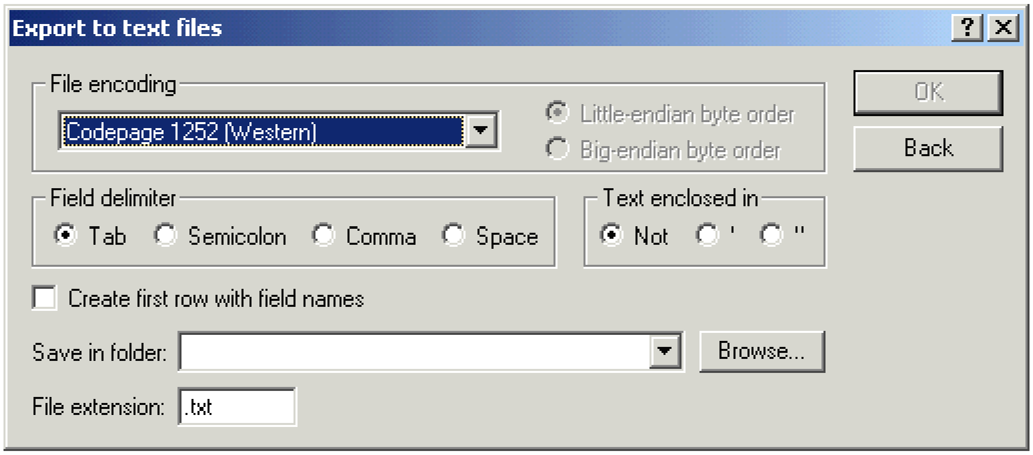
Create text fields with length: 255

The **Build** button lets you create the necessary ADO connection string - for more information see the Import Database command and ADO connection string.

Export to Text File

This command button allows you to **specify** the formatting of the text files to be exported.

If you are exporting XML data to text files, you must specify the desired character-set encoding to be used. The same options are also available in the Import text file dialog box.



Export to text files

File encoding

Codepage 1252 (Western)

● Little-endian byte order

○ Big-endian byte order

Field delimiter

● Tab ○ Semicolon ○ Comma ○ Space

Text enclosed in

● Not ○ ' ○ "

□ Create first row with field names

Save in folder:

File extension: .txt

The set of text files will be generated in the folder you specify, and each file name will be generated from the corresponding element name. You can also specify the file extension to be used.

11.9.8 Tamino

Software AG and Altova, Inc. have entered into a worldwide agreement giving XMLSpy users easy and low-cost access to Tamino. The partnership also extends Software AG's reach to Altova's community of developers.

Under the terms of the agreement, Altova's XMLSpy users and Software AG's Tamino users obtain an integrated product bundle. Tamino XML Server is available on multiple platforms from PC to Mainframe.

XMLSpy and Tamino (limited edition) are available for download and purchase at order page. The Tamino database capacity varies from 50MB (per XMLSpy Enterprise/Professional single license) up to 1000 MB depending on the number of XMLSpy licenses purchased.

Having paid for the bundle, you will receive an e-mail with details on how and where you can download Tamino.

Installing Tamino

Having downloaded the XMLSpy and Tamino bundle you are ready to start the installation process.

General installation process:

1. Download and install XMLSpy.
2. Download the Tamino XML Server.
3. Install Apache server (if necessary).
4. Install Tamino Server 3.1.1.
5. Create a database in Tamino.
6. Use XMLSpy.

Specific Tamino installation instructions are provided in HTML format. Open the Tamino documentation home page located at:

<Root>/Windows/INO/Docu/overview.htm

where "<**Root**>" is the directory where you unpacked Tamino, and select the entry "Installing Tamino".

Scope of this documentation:
This documentation assumes that you have successfully installed both XMLSpy Enterprise or Professional edition, and Tamino Server 3.1.1.

Installing WebDAV

Installing and using WebDAV:
XMLSpy supports WebDAV. Installing the WebDAV server is **optional**. Using the Tamino functionality provided by XMLSpy is not compatible with WebDAV, due to limitations in keeping repositories synchronized in both Tamino Server and WebDAV.

Installing WebDAV:

- Install WebDAV server from Software AG.
- Create a WebDAV store, and restart WebDAV server.

Tamino Schema Extensions

While XML Schema covers the logical aspects of document type definitions, it does not prescribe a way how to define the physical aspects. Many XML processors require extra information on how to process the instances of a given document class.

Tamino, for example, requires information as to which collection a document class belongs, which elements are used for indexing, what type of indexing is used, how document elements may be mapped onto fields in external databases, and so on.

For these and other purposes XML Schema provides an extension mechanism. Any schema, element or attribute definition in XML Schema can be equipped with one or more annotations.

Each **annotation** can consist of two child elements: **documentation** and **appinfo**. The documentation element contains documentation for human readers, while the appinfo element contains information for machines.

Tamino uses this **appinfo** element to store Tamino-related information within a schema file:

```
<?xml version = "1.0" encoding = "UTF-8"?>
<xs:schema xmlns:xs = "http://www.w3.org/2001/XMLSchema"
           xmlns:tsd =
    "http://namespaces.softwareag.com/tamino/TaminoSchemaDefinition">
  <xs:annotation>
    <xs:appinfo>
      <tsd:schemaInfo name = "album">
        <tsd:collection name = "encyclopedia"/>
        <tsd:doctype name = "album">
          <tsd:logical>
            <tsd:content>open</tsd:content>
            <tsd:accessOptions>
              <tsd:read/>
              <tsd:insert/>
              <tsd:delete/>
              <tsd:update/>
            </tsd:accessOptions>
          </tsd:logical>
        </tsd:doctype>
      </tsd:schemaInfo>
    </xs:appinfo>
  </xs:annotation>
...
</xs:schema>
```

These annotations are generated by the Tamino Schema Editor, as well as in XMLSpy Schema/WSDL Design view when working with Tamino Extensions.

Creating, Saving, and Listing Tamino Schemas

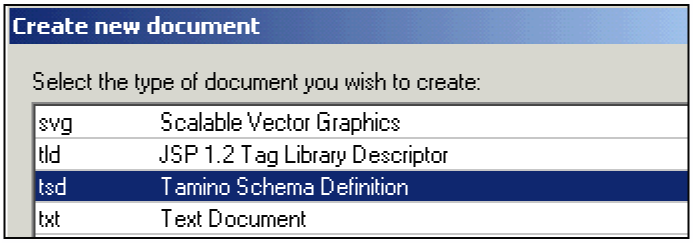
All communication between Tamino and XMLSpy occurs using the HTTP communication protocol.

Populating the Tamino database

The first prerequisite to be able to save any data in Tamino, is to create and save a schema to Tamino. In the following steps a simple schema will be created. The same schema is also attached in the Examples folder as `BMW.tsd`.

Creating a Tamino Schema Definition file:

1. Create a schema using the menu option **File | New** in XMLSpy, and select the **tsd Tamino Schema Definition** entry.

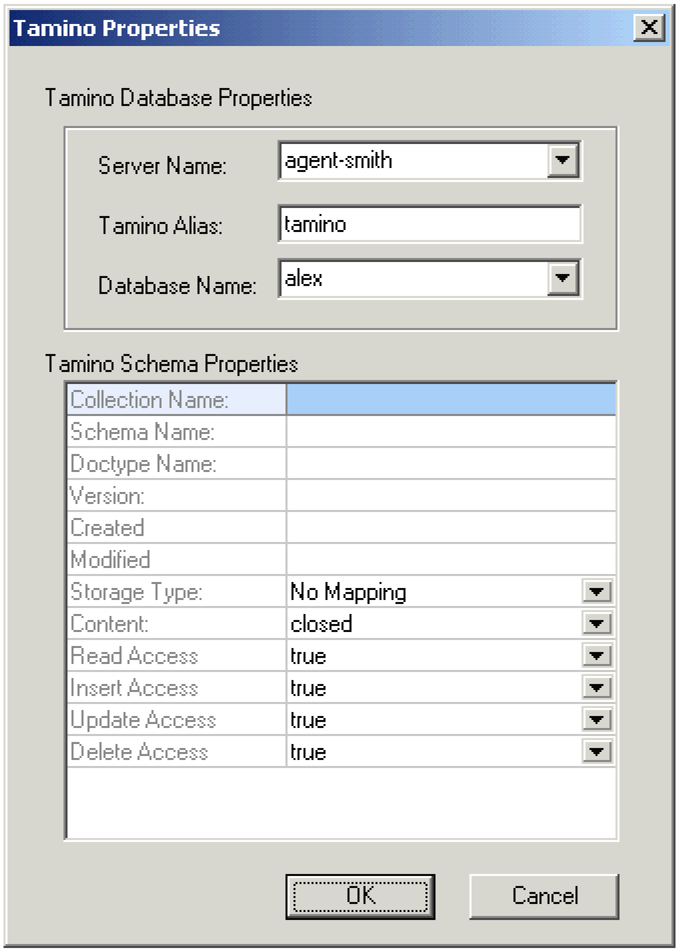


Create new document

Select the type of document you wish to create:

svg	Scalable Vector Graphics
tld	JSP 1.2 Tag Library Descriptor
tsd	Tamino Schema Definition
txt	Text Document

This opens the Tamino Properties dialog box where you define the database properties.



Tamino Properties

Tamino Database Properties

Server Name: agent-smith

Tamino Alias: tamino

Database Name: alex

Tamino Schema Properties

Collection Name:	
Schema Name:	
Doctype Name:	
Version:	
Created	
Modified	
Storage Type:	No Mapping
Content:	closed
Read Access	true
Insert Access	true
Update Access	true
Delete Access	true

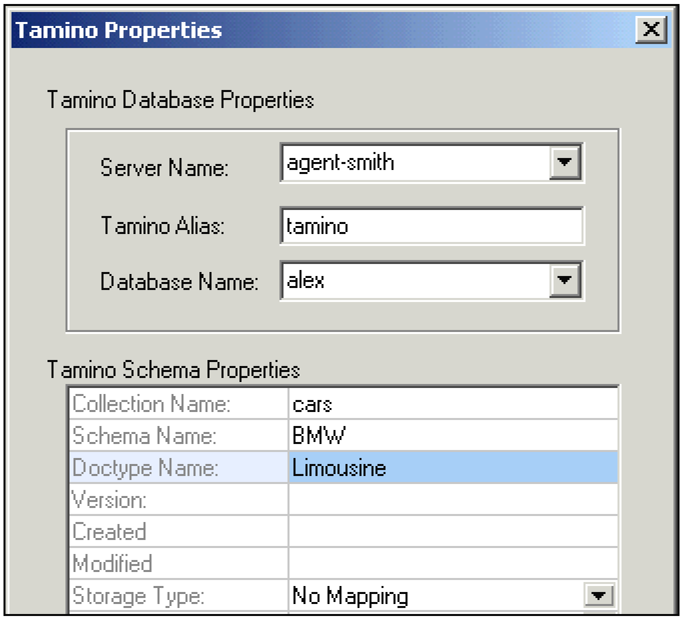
If not already supplied, enter the **Server** and **Database** and **Tamino Alias** names in the appropriate fields.

2. Enter the **Collection Name**, **Schema Name** and **Doctype name** in the respective fields, and confirm with **OK**. Entering these data here creates them in Tamino.

Collection Name:	cars
Schema Name:	BMW
Doctype Name:	Limousine

Please note:

The Tamino Alias is configured by the database administrator, and is sometimes used for security purposes, or if aliases are already occupied. Please contact your database administrator for more information on the Tamino Alias.



Tamino Properties

Tamino Database Properties

Server Name: agent-smith

Tamino Alias: tamino

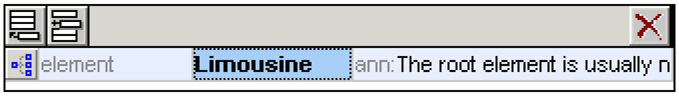
Database Name: alex

Tamino Schema Properties

Collection Name:	cars
Schema Name:	BMW
Doctype Name:	Limousine
Version:	
Created	
Modified	
Storage Type:	No Mapping

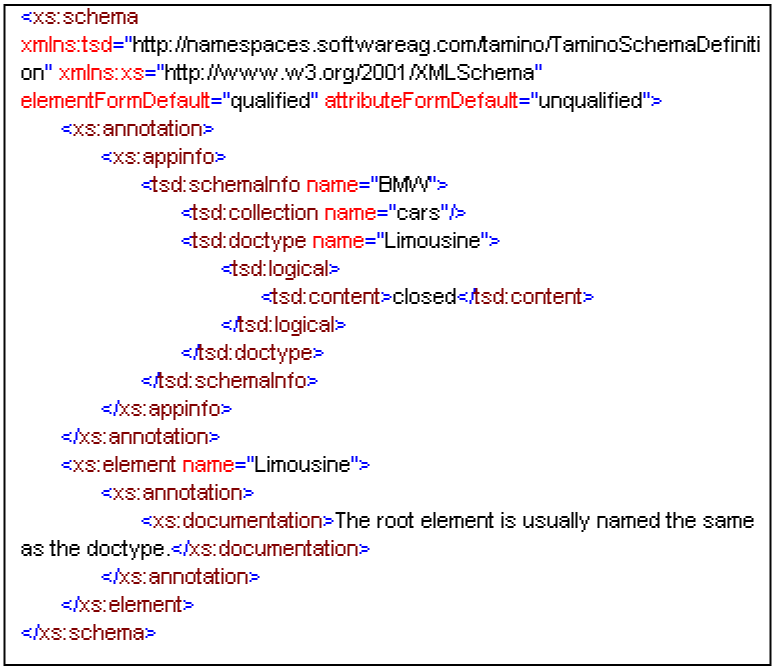
This creates a Tamino schema definition file which is displayed in the Schema/WSDL Design view.

3. Enter the Doctype Name `Limousine` in the Root element field. (The Doctype name can be any global element, it does not have to be the root element.) This Doctype name must, however be present in the schema file, in order to save it in Tamino.



element **Limousine** ann: The root element is usually n

4. This completes the definition stage of making the schema file conformant to the Tamino database. A **Tamino** tab has also been added to the Details entry helper. The Tamino extensions can be applied via the Details entry helper to Elements and Attributes. Clicking the **Text** tab displays the schema definitions.



```
<xs:schema
xmlns:tsd="http://namespaces.softwareag.com/tamino/TaminoSchemaDefiniti
on" xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
	<xs:annotation>
		<xs:appinfo>
			<tsd:schemaInfo name="BMW">
				<tsd:collection name="cars"/>
				<tsd:doctype name="Limousine">
					<tsd:logical>
						<tsd:content>closed</tsd:content>
					</tsd:logical>
				</tsd:doctype>
			</tsd:schemaInfo>
		</xs:appinfo>
	</xs:annotation>
	<xs:element name="Limousine">
		<xs:annotation>
			<xs:documentation>The root element is usually named the same
as the doctype.</xs:documentation>
		</xs:annotation>
	</xs:element>
</xs:schema>
```

5. Design your schema with the schema editor as follows:

   Add to the root element "Limousine" a sequence node and then add the following children element nodes:

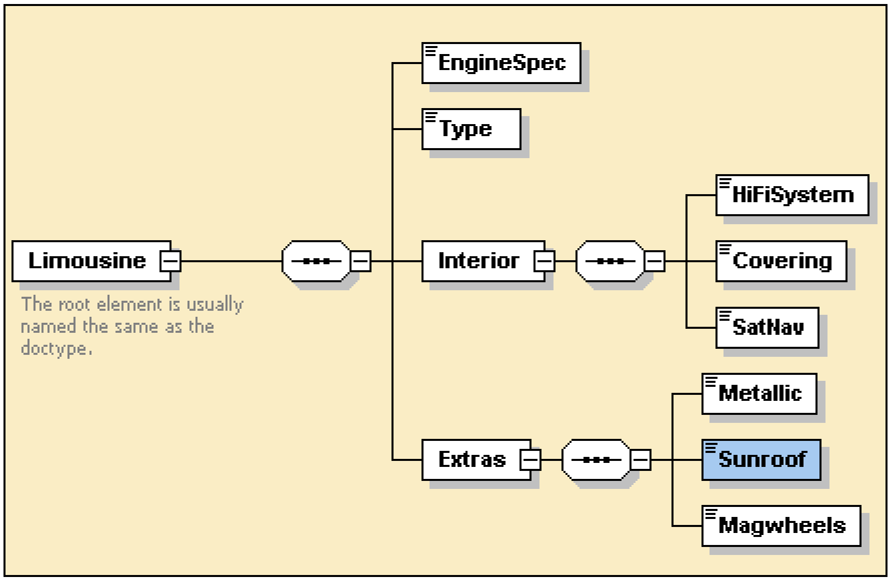
EngineSpec	xs:string
Type	xs:string
Interior	
Extras	

   Append a sequence node to the "Interior" element and add the following children:

HiFiSystem	xs:string
Covering	xs:string
SatNav	xs:string

   Append a sequence node to the "Extras" element and add the following children:

Metallic	xs:boolean
Sunroof	xs:boolean
Magwheels	xs:boolean

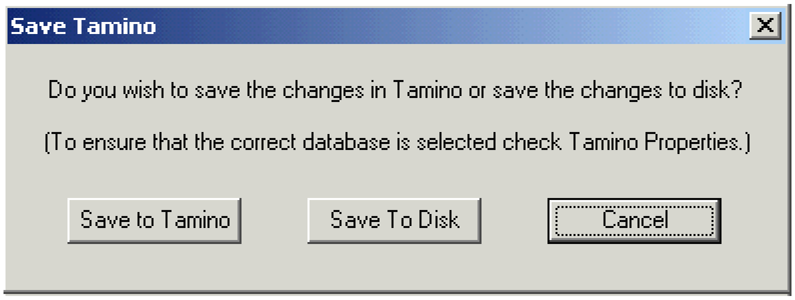


Please note:

Tamino does not support **Global complex** types!

Saving a Tamino schema:

- Press **CTRL**+S, and then click the **Save to Tamino** button.
  This saves the schema to the Tamino database. You can also select the menu option **Convert | Tamino | Save to Tamino...** to save the schema to the database directly.


Save Tamino

Do you wish to save the changes in Tamino or save the changes to disk?

(To ensure that the correct database is selected check Tamino Properties.)

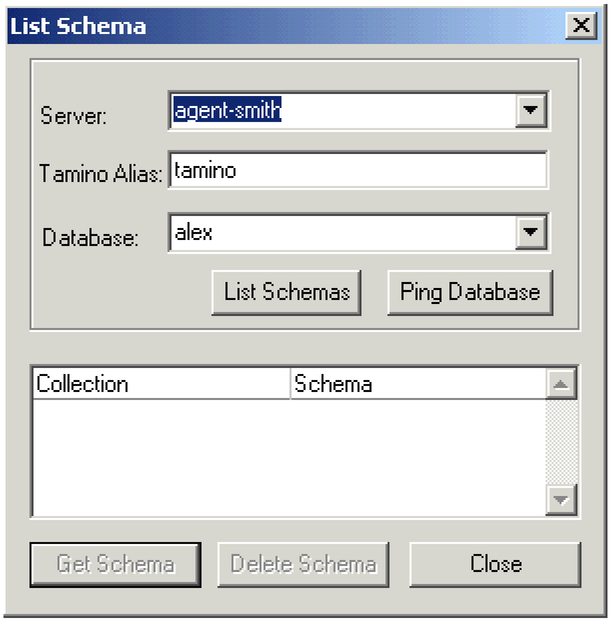
Please note:
Saving to Tamino invokes the **Tamino schema validator**. This validator takes Tamino extensions and other Tamino-specific rules into consideration; it is not the same as the XMLSpy validator. If the schema is not valid according to the Tamino validator, an error message appears describing the problem. Please see the Tamino documentation if this is the case.

Saving to Disk
Saving the schema using the above method, places it in the database without having saved it on the hard disk. The main window tab contains **Untitledxx.xsd** as the schema name. To save the schema **locally**, click the **Save To Disk** button.

If you want to create XML files in XMLSpy based on schemas saved in Tamino, make sure that you use the **Save to Disk** function, to save a copy of the schema locally.

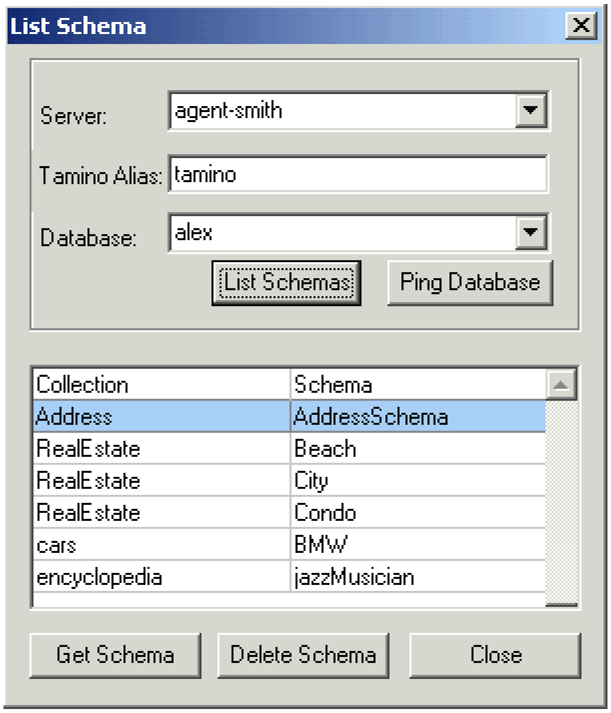
Listing schemas in the Tamino database:

1. Select the menu option **Convert | Tamino | List Schema...**
   This opens the List Schema dialog box.



Server: agent-smith
Tamino Alias: tamino
Database: alex

2. If not already supplied, enter the Server and Database names in the appropriate fields.
3. Click the **List Schemas** button.
   A list of schemas, and associated Collections, of the current database are displayed in the list box.



Server: agent-smith
Tamino Alias: tamino
Database: alex

Collection	Schema
Address	AddressSchema
RealEstate	Beach
RealEstate	City
RealEstate	Condo
cars	BMW
encyclopedia	jazzMusician

Please note: The **Ping Database** button allows you to test the connection to Tamino.

To open a schema:

- Double-click the schema name in the list box, or
- Mark the respective schema, and click the Get Schema button.

To delete a schema from Tamino:

1. List the schemas in the List schema dialog box.
2. Mark the schema you want to delete, and click the **Delete Schema** button.

Validating Tamino schemas

- Having retrieved a schema from Tamino, select the menu option **XML | Validate**, or press **F8**.
  The XMLSpy validator is used in this case; XMLSpy error messages appear if the schema is invalid.

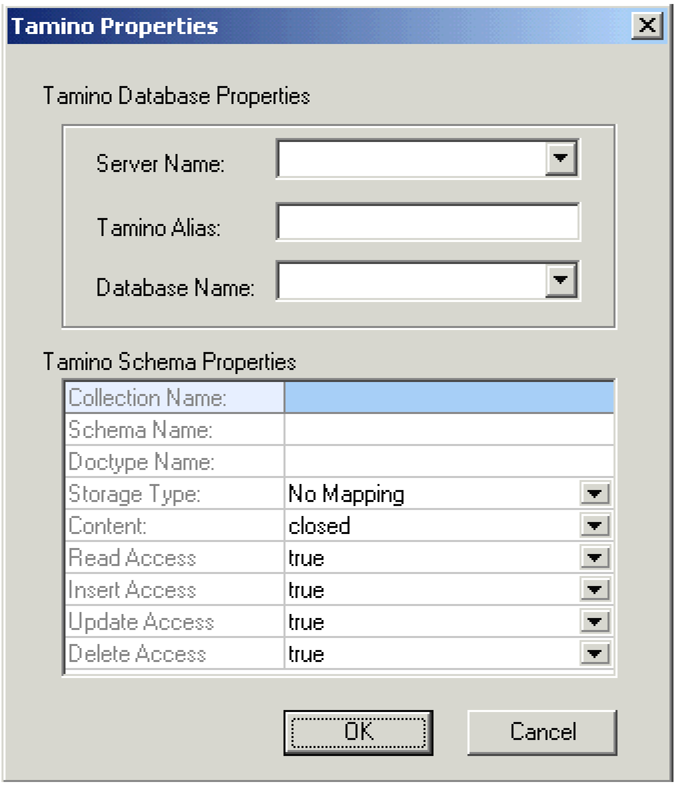
However, **saving** the schema back **to Tamino** invokes the Tamino schema validator. This validator takes Tamino extensions and other Tamino specific rules into consideration; it is not the same as the XMLSpy validator. If the schema is not valid according to this validator, an error message appears detailing the problem. Please see the Tamino documentation if this is the case.

Adding Legacy Schemas to Tamino

Schemas that have been created in XMLSpy can be easily added to Tamino Server.

To add an existing schema to Tamino Server:

1. Open the schema in the Schema/WSDL Design view.
2. Select the menu option **Schema design | Enable Tamino Schema Extensions**.
   A dialog box opens if you are using a schema prefix other than 'xs'. Tamino requires that the namespace prefix be 'xs'.
3. Click **OK** if the dialog box opened.
   A further dialog box is opened, asking if the file extension should be changed to `.tsd`. This is optional. (Click **OK** to confirm.)
4. The Tamino Properties dialog box is automatically opened. Enter the database properties, as well as the schema properties: Collection, Schema, and Doctype names and confirm with **OK**.



Collection Name:	
Schema Name:	
Doctype Name:	
Storage Type:	No Mapping
Content:	closed
Read Access	true
Insert Access	true
Update Access	true
Delete Access	true

5. Select the menu option **Convert | Tamino | Save to Tamino.**
   A confirmation message appears when the schema is successfully saved.

Tamino Schema Definition Limitations

Limitations of the Tamino Schema Definition Language Implementation (Copyright Software AG 2001)
The Tamino schema definition language as implemented in the current version of Tamino has the following limits:

Logical TSD3 restrictions not deducible by syntactical subset of logical W3C XS meta schema:

- per model group (i.e. the content of 1 complexType) the element names of each local element and all referenced global elements must be unique. That means for each element name either one local element or several references to global elements are allowed.
- the xs:type and xs:base attribute allows only datatypes defined in XML schema part 2, also see commented definition of tsd:native Datatype in this documentation.
  Logical TSD3 restrictions compared to full DTD capabilities, which are expressable in full XML schema (e.g. General Entities)
- enumerations for elements without subelements and with attributes cannot be specified (restriction of XML schema: this cannot not be expressed without named complex types)
  Physical TSD3 restrictions
- attachment of physical schema definition not possible without tsd:which element below recursive elements
- only one schemaInfo, elementInfo, attributeInfo element possible for each schema, element or attribute-definition

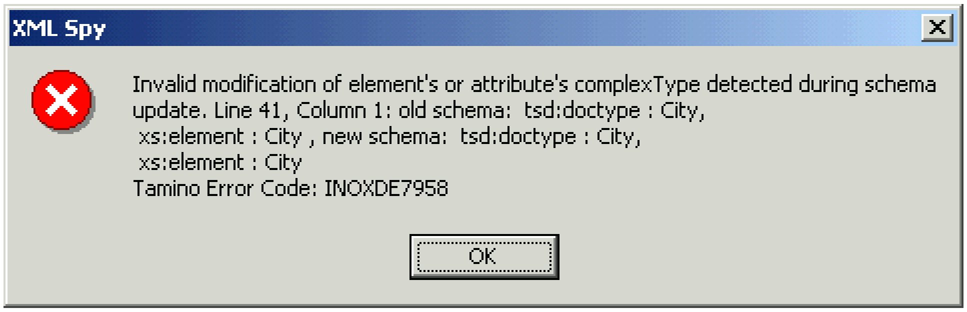
Limitations of the XMLSpy and Tamino bundle

- Transaction oriented processing not supported
- Only XML documents and XSD Schema documents can be saved in Tamino via XMLSpy
- Including of schemas within schemas is not supported
- Importing of schemas is not supported
- Namespaces (prefixes) in a schema are not supported by Tamino Server
- Using Tamino functionality provided by XMLSpy is not compatible with WebDAV

Editing Tamino Schemas

Editing schemas (which reference XML documents in the database)
Changing a schema definition on which multiple XML documents are based, might invalidate all associated XML documents. On saving, Tamino checks to see if the schema has been changed inappropriately. A Tamino error message appears if this is the case.

E.g. A schema and associated document exist in Tamino. The schema is opened in XMLSpy and a new mandatory element is added to a complex type. Saving this schema back to Tamino causes an error message to appear.



Invalid modification of element's or attribute's complexType detected during schema update. Line 41, Column 1: old schema: tsd:doctype : City,
xs:element : City , new schema: tsd:doctype : City,
xs:element : City
Tamino Error Code: INOXDE7958

Changing the occurrence of the mandatory element to optional (min=0 and max=1), allows the schema to be saved back to Tamino without invalidating any of the referenced XML documents.

Creating XML Files Based on Tamino Schemas

Creating an XML file based on a Tamino schema:

You can create XML files based on schemas in the following ways:

- By using an existing XML document as a **template**, where both the XML and schema file exist in Tamino, or
- By using a schema file saved in Tamino, as a basis for the new XML document.

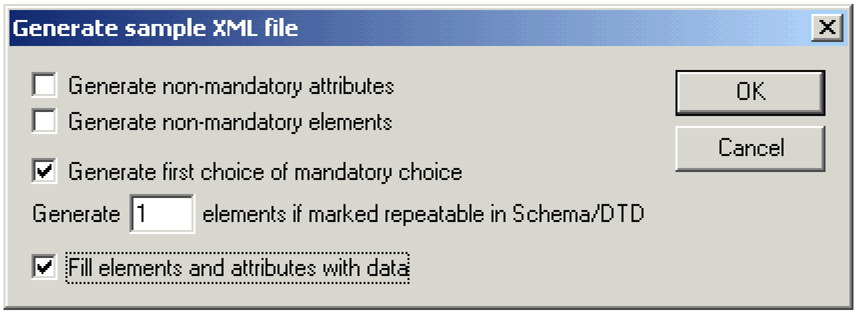
Creating an XML file based on another XML (template) file:

1. Create a query for the XML document you want to use as a template, and retrieve the XML document from Tamino Server. (You can also use the alternative Get File method to retrieve the XML file.)
2. Edit the XML document.
3. Use the menu option **Convert | Tamino | Add Document**, to save the new XML document back to Tamino Server.
   Please note: There are sample XML Files available in the `Examples` folder, `cars1.xml` and `cars2.xml`.

Creating an XML file based on a Tamino schema file (saved locally):

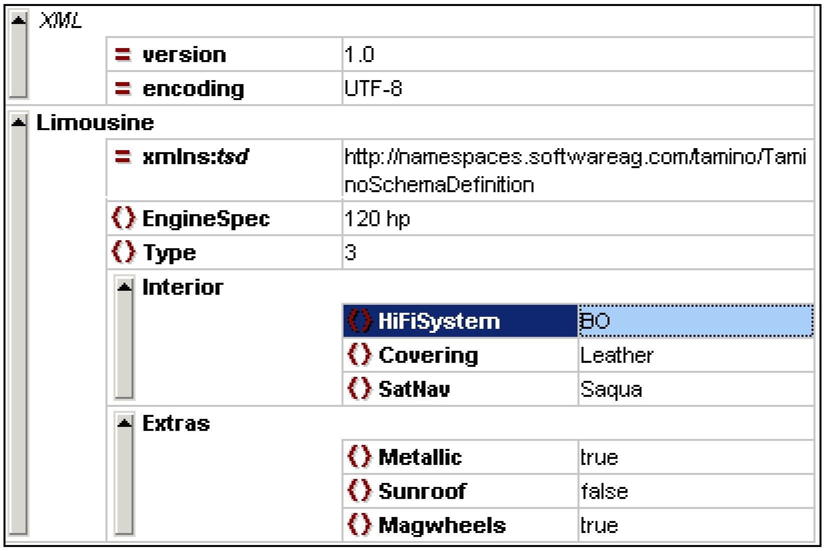
Having opened a schema file saved in Tamino (this is the BMW schema in our example), then

1. Make sure the opened schema file is the active document.
2. Select **Convert | Tamino | Generate Sample XML file**.



□ Generate non-mandatory attributes
□ Generate non-mandatory elements
☑ Generate first choice of mandatory choice
Generate 1 elements if marked repeatable in Schema/DTD
☑ Fill elements and attributes with data

3. A dialog appears. Ensure that the "Generate first choice of mandatory choice", and "Fill elements and attributes with data" check boxes are selected. Confirm with **OK**.
4. A sample XML file is generated. Replace the data as shown below.



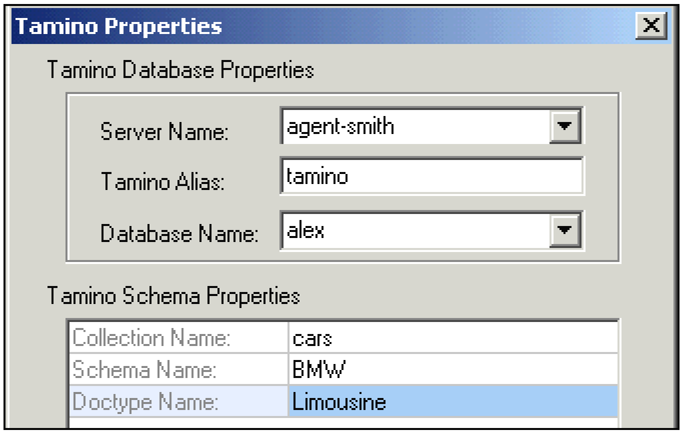
XML			
	= version	1.0	
	= encoding	UTF-8	
Limousine			
	= xmlns:*tsd*	http://namespaces.softwareag.com/tamino/TaminoSchemaDefinition	
	{} EngineSpec	120 hp	
	{} Type	3	
	Interior		
		{} HiFiSystem	BO
		{} Covering	Leather
		{} SatNav	Saqua
	Extras		
		{} Metallic	true
		{} Sunroof	false
		{} Magwheels	true

Please note:

When you select the menu option **XML | Validate**, the XML file is validated against the **local** Tamino schema file using the XMLSpy validator.

Adding an XML file / record / document, to Tamino

1. Select the menu option **Convert | Tamino | Add Document** to add the active XML document to Tamino.
   A prompt appears asking for more information.
2. Enter the Collection, Schema, and Doctype names in the Tamino properties dialog box, and confirm with **OK**.



Tamino Database Properties

Server Name: agent-smith

Tamino Alias: tamino

Database Name: alex

Tamino Schema Properties

Collection Name:	cars
Schema Name:	BMW
Doctype Name:	Limousine

   The XML file is now associated to the schema saved in Tamino. A confirmation message appears when the XML document has been saved successfully.
3. Change one of the entries in the currently open XML file (metallic=false), and select the menu entry **Convert | Tamino | Add Document** again.
   This adds the edited XML file to the database using the previously defined Tamino settings. You now have two XML files associated to the cars Collection, BMW schema, and Limousine Doctype.

Please note:

Validating an **XML** file that has been **added to Tamino** Server, validates it against the referenced schema file in Tamino.

Only XML and associated XSD files can currently be saved in Tamino via XMLSpy.

Querying and Retrieving Tamino Data

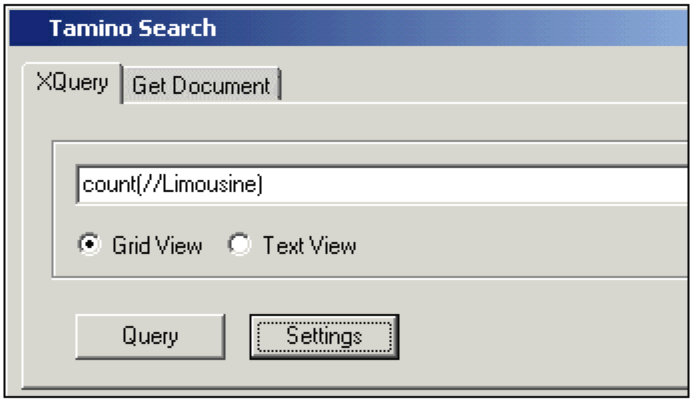
Querying the Tamino database

A query is defined by using the menu option **Convert | Tamino | Search**. Tamino uses a query language called **XQuery**. The syntax of this language follows the standard XPath notation.

You can create an XQuery as well as retrieve a specific file from the database (if you know the specific record ID) using this dialog box. The query result is an XML document that appears in either the Grid or Text view.

E.g
we want to find out how many records exist with the Doctype "Limousine". We will use the XPath command **count()** as the query statement.

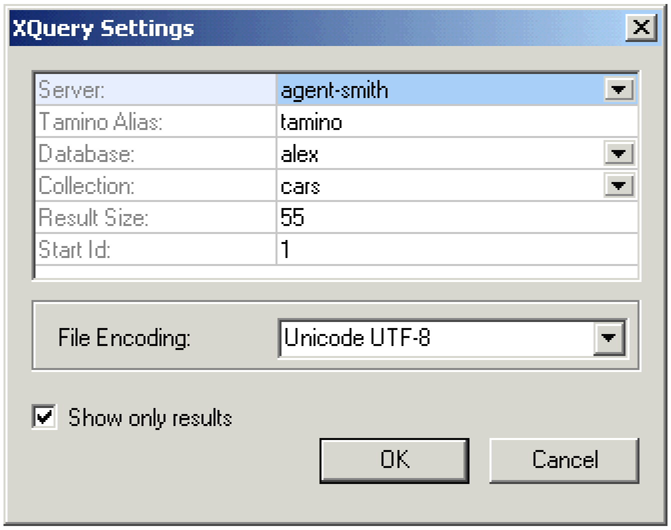
1. Enter the **count(//Limousine)** query in the XQuery field.



count(//Limousine)

● Grid View ○ Text View

2. Having defined the query you also have to define which Database, Collection and Doctype have to be searched. Click the **Settings** button to enter these settings, and confirm with **OK**. Please note that Tamino is **case sensitive**, so make sure you enter the data correctly.



Server:	agent-smith
Tamino Alias:	tamino
Database:	alex
Collection:	cars
Result Size:	55
Start Id:	1

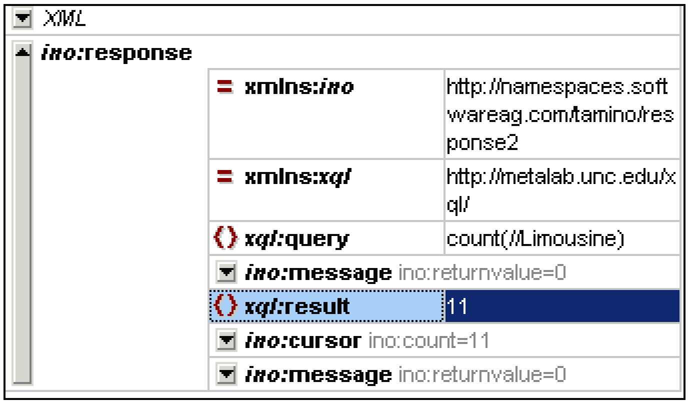
File Encoding: Unicode UTF-8

☑ Show only results

3. Click the **Query** button to execute the query, and then **Close**, to close the dialog box.

The query result is an XML document with the **xql:result** field displaying the result (11 in this case).

Please note: The default Result Size field is 5 results; it has been changed to 55 in this example. Activating the "Show only results" check box causes the query result document to contain only result data.



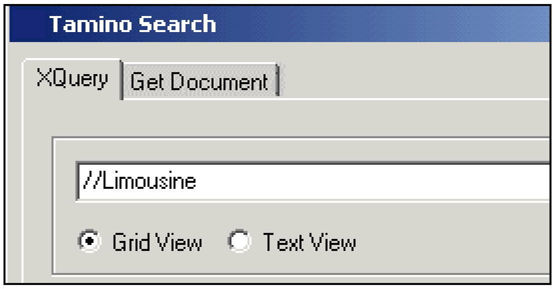
XML

ino:response

= xmlns:ino	http://namespaces.softwareag.com/tamino/response2
= xmlns:xql	http://metalab.unc.edu/xql/
() xql:query	count(//Limousine)
ino:message ino:returnvalue=0	
() xql:result	11
ino:cursor ino:count=11	
ino:message ino:returnvalue=0	

Retrieving specific documents from Tamino:

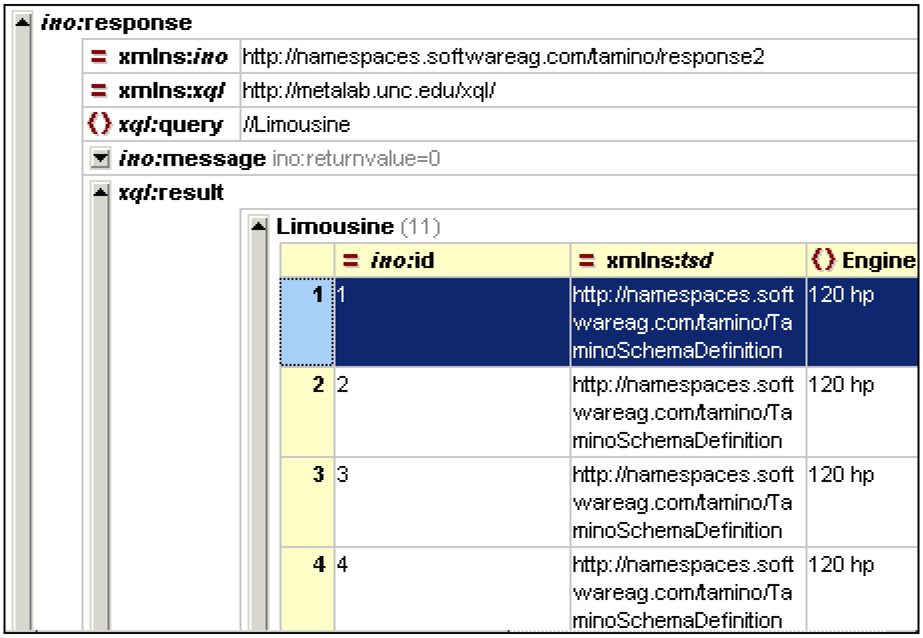
1. Define a new query, which should supply all XML documents of Doctype Limousine, and click the **Query** button. Keep the same XQuery settings as before.



//Limousine

● Grid View ○ Text View

The resulting query response is opened, multiple results are displayed in Table view.



ino:response

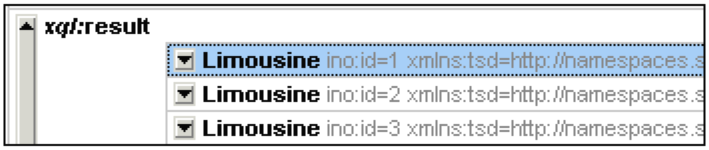
= xmlns:ino	http://namespaces.softwareag.com/tamino/response2
= xmlns:xql	http://metalab.unc.edu/xql/
() xql:query	//Limousine
ino:message ino:returnvalue=0	

xql:result

Limousine (11)

	= ino:id	= xmlns:tsd	() Engine
1	1	http://namespaces.softwareag.com/tamino/TaminoSchemaDefinition	120 hp
2	2	http://namespaces.softwareag.com/tamino/TaminoSchemaDefinition	120 hp
3	3	http://namespaces.softwareag.com/tamino/TaminoSchemaDefinition	120 hp
4	4	http://namespaces.softwareag.com/tamino/TaminoSchemaDefinition	120 hp

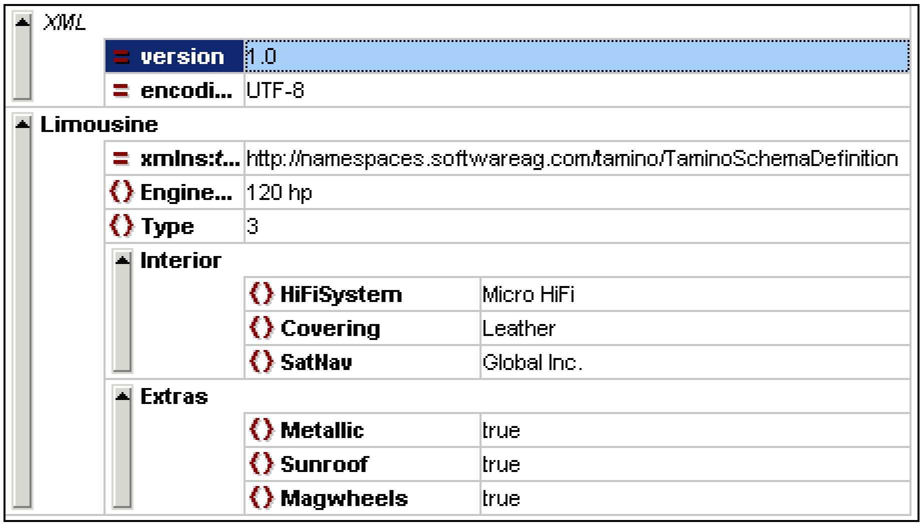
2. Click the table row number in the query result document, and select **Convert | Tamino | Get active Document**. If the table view is not active, select the respective Limousine row.



xql:result

Limousine	ino:id=1 xmlns:tsd=http://namespaces.s
Limousine	ino:id=2 xmlns:tsd=http://namespaces.s
Limousine	ino:id=3 xmlns:tsd=http://namespaces.s

The query result appears as an XML document in the main window.



XML

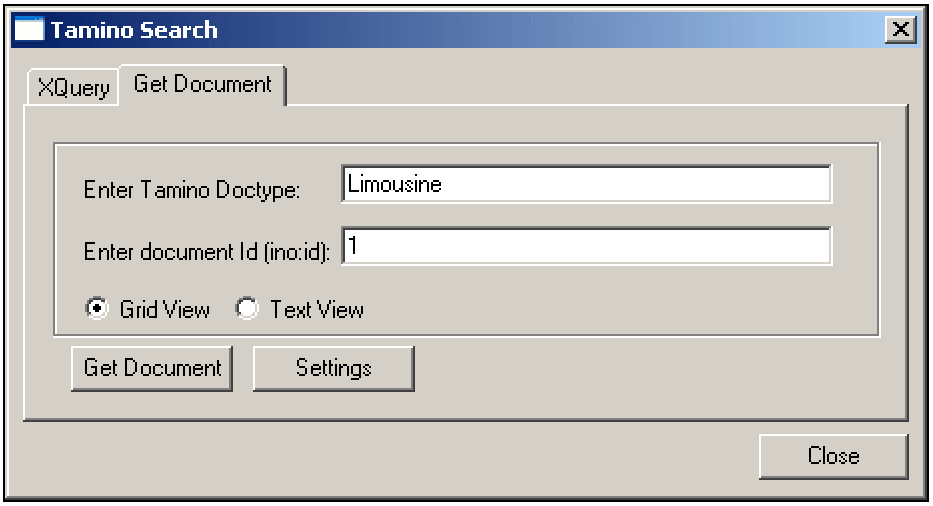
version	1.0
encodi...	UTF-8

Limousine

xmlns:t...	http://namespaces.softwareag.com/tamino/TaminoSchemaDefinition	
Engine...	120 hp	
Type	3	
Interior	HiFiSystem	Micro HiFi
	Covering	Leather
	SatNav	Global Inc.
Extras	Metallic	true
	Sunroof	true
	Magwheels	true

Get Document method of retrieving a document:

1. Note the ID number from the query document, and open the Query dialog box again. The document ID is the **ino:id=x** entry.
2. Click the **Get Document** tab, and enter the Tamino Doctype in the Enter Tamino Doctype text box.
3. Enter the document ID in the Enter document Id text box.



Tamino Search

Enter Tamino Doctype: Limousine

Enter document Id (ino:id): 1

● Grid View ○ Text View

4. Click the **Get Document** button to open the XML document.

To save a Tamino query:

- Use the **CTRL+S** shortcut to save the query document locally.
  This saves the query file as an XML document; it is not linked to Tamino Server in any way.

Updating a query:
Having defined and executed a query once, you can reuse the query document to execute the same query.

1. Switch back to the original query document.
2. Select the menu option **Convert | Tamino | Refresh**.
   This queries Tamino with the same settings as before, and delivers a new query result document containing updated data.

Editing existing XML documents and adding them to Tamino
Change some of the data in the XML document and select the menu option **Convert | Tamino | Add Document**. This saves the edited file back to Tamino. A confirmation message appears when the file is added successfully.

11.9.9 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

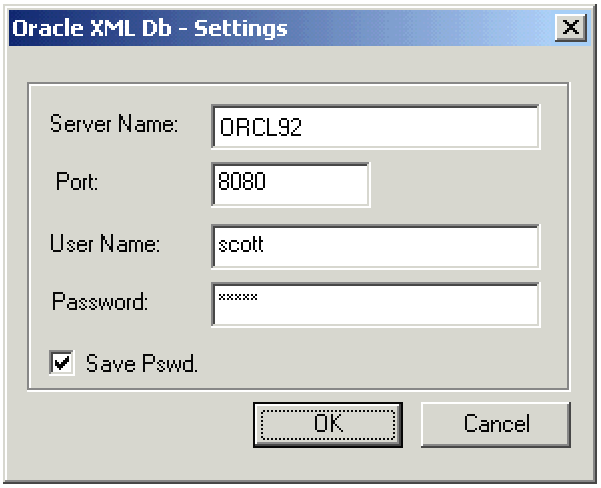
An Oracle XML DB Demo installation manual is provided at http://otn.oracle.com/sample_code/tech/xml/xmldb/XDBBasicDemo.pdf. The document describes how to install the Oracle software as well as how to configure it to work with XMLSpy.

Search...

🔍

The **Convert | Oracle XML DB | Search** command lets you query the Oracle XML database using DBUri statements.

When you select this command, the following dialog appears:



Server Name: ORCL92

Port: 8080

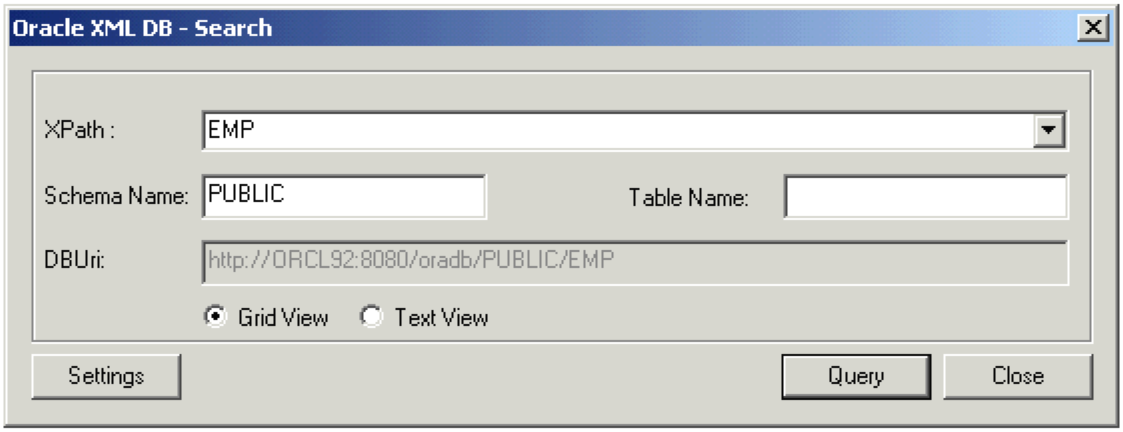
User Name: scott

Password: *****

☑ Save Pswd.

Enter your Oracle settings and click **OK**.

Enter the query as an XPath in the **XPath** field. The query you enter is appended to the DBUri field below, and is sent to the server using the http protocol.



XPath : EMP

Schema Name: PUBLIC

Table Name:

DBUri: http://ORCL92:8080/oradb/PUBLIC/EMP

● Grid View ○ Text View

The **Schema Name** field must be entered manually for the very first query; it is then automatically filled in on successive queries.

The **Table Name** field is an optional table that you can query.

Settings
The **Settings** button opens the Settings dialog box in which you define the query settings: Server Name, Port, User Name and Password.

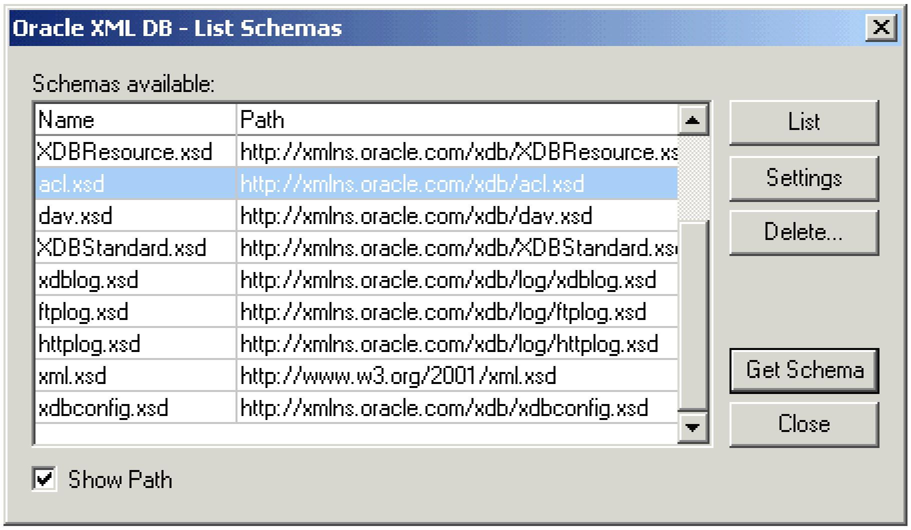
Query
The **Query** button sends off the query and closes the dialog box.

The **query result** appears in either the Enhanced Grid or Text view depending on the option you select in the dialog box. If the query was not successful an error message appears.

List schemas

To list the schemas available in the Oracle database:

1. Select the menu option **Convert | Oracle XML Db | List Schemas**.
   This opens the Oracle XML DB - List schemas dialog box.
2. Click the **List** button to list the schemas in the current database.



Name	Path
XDBResource.xsd	http://xmlns.oracle.com/xdb/XDBResource.xs
acl.xsd	http://xmlns.oracle.com/xdb/acl.xsd
dav.xsd	http://xmlns.oracle.com/xdb/dav.xsd
XDBStandard.xsd	http://xmlns.oracle.com/xdb/XDBStandard.xs
xdblog.xsd	http://xmlns.oracle.com/xdb/log/xdblog.xsd
ftplog.xsd	http://xmlns.oracle.com/xdb/log/ftplog.xsd
httplog.xsd	http://xmlns.oracle.com/xdb/log/httplog.xsd
xml.xsd	http://www.w3.org/2001/xml.xsd
xdbconfig.xsd	http://xmlns.oracle.com/xdb/xdbconfig.xsd

3. Double-click one of the schemas in the schema list, or click one of the schemas and click the **Get Schema** button.
   The selected schema appears in the Schema/WSDL Design view.

The Oracle XML DB - List schemas dialog box allows you to select further options.

Settings
This button allows you to define or change the current Oracle XML Db settings.

Delete
This button deletes a registered schema (visible in the list box) from Oracle XML Db.

1. Click **Delete**.
2. Select either the ADO or ODBC connection string radio button, and click the **Build** button.
   This opens the Data Link properties dialog box (or Select Data source dialog box, if you selected ODBC).
3. Select the **MS OLE DB Provider for Oracle** and click the **Next** button.
4. Enter the server details in the respective fields: server name, User name, Password, and click **OK**.
   The ADO connection string is now visible in the connection string field.
5. Click the **Delete** button to delete the schema.

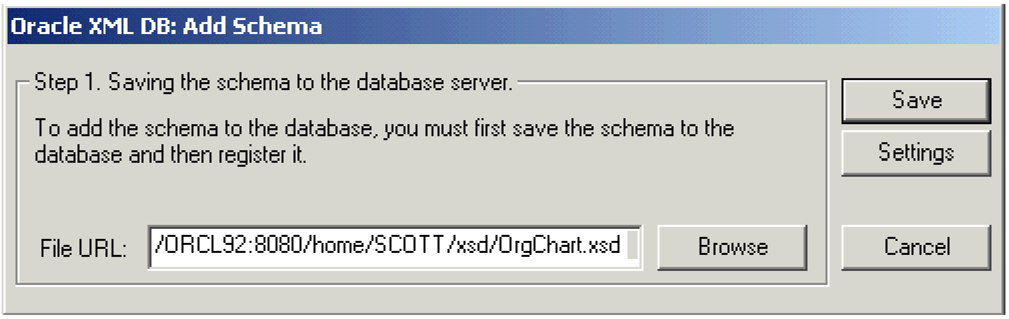
Get Schema
Opens the currently marked schema in the Schema/WSDL Design view.

Close
Closes the dialog box.

Add Schema

This command allows you to add an XML schema to Oracle XML Db. To add an XML document instance file to the database, please see Adding XML document instances.

1. Select the menu option **Convert | Oracle XML Db | Add Schema**.
   This is the first step in the add schema process, the schema is saved to the database.
   The File URL is automatically entered in the **File URL** field.
   Click the Browse button to define where you want to save the file. Enter the file name of the new schema file.

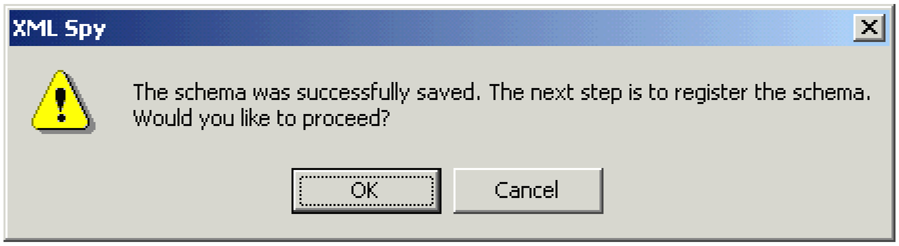

Oracle XML DB: Add Schema

Step 1. Saving the schema to the database server.

To add the schema to the database, you must first save the schema to the database and then register it.

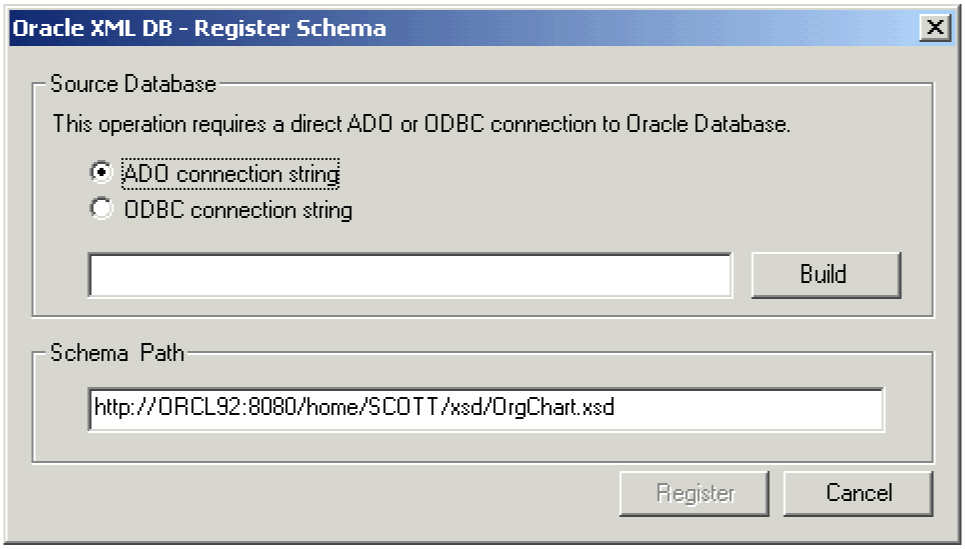
File URL: /ORCL92:8080/home/SCOTT/xsd/OrgChart.xsd

2. Click the **Save** button to save the schema to the database. A message appears telling you that the schema was saved successfully. Click **OK** to proceed with the registration process.


XML Spy

The schema was successfully saved. The next step is to register the schema. Would you like to proceed?

   This opens the **Register Schema** dialog box. The path of the schema you just saved is visible in the **Schema Path** field.
3. Select either the ADO or ODBC connection string radio button, and click the **Build** button.


Oracle XML DB - Register Schema

Source Database

This operation requires a direct ADO or ODBC connection to Oracle Database.

● ADO connection string
○ ODBC connection string

Schema Path

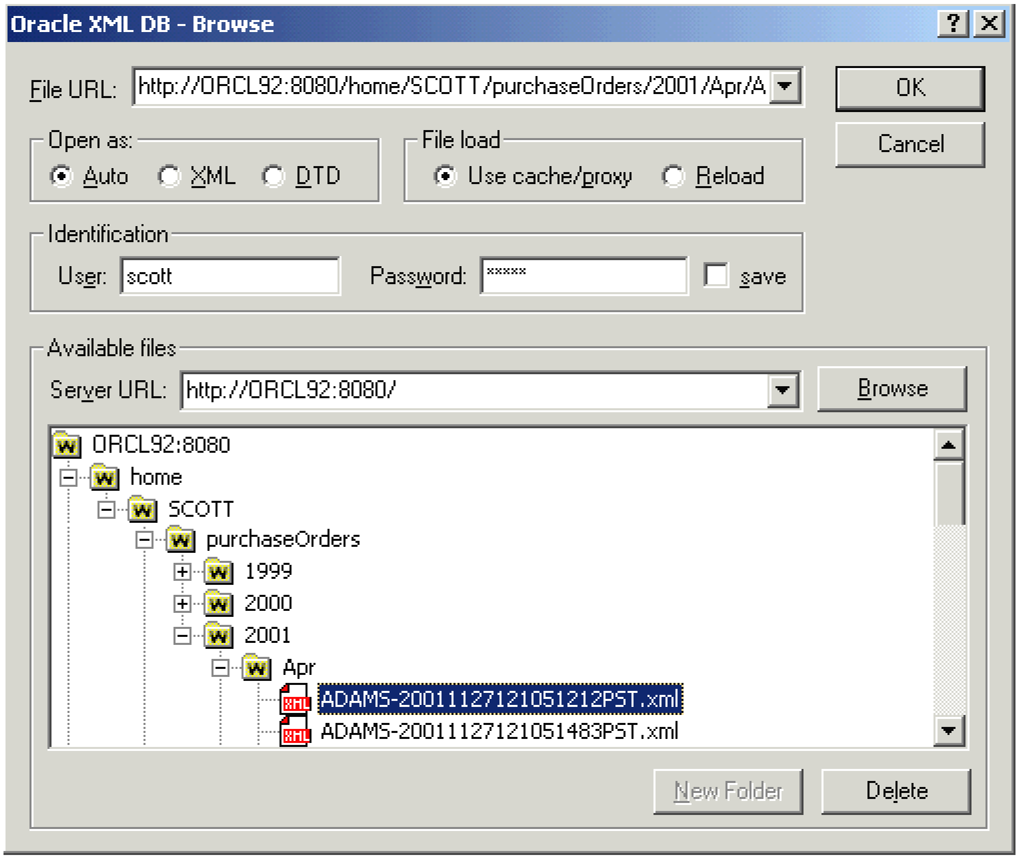
http://ORCL92:8080/home/SCOTT/xsd/OrgChart.xsd

   This opens the Data Link properties dialog box (or Select Data source dialog box, if you selected ODBC).

4. Select the **MS OLE DB Provider for Oracle** and click **Next**.
5. Enter the server details in the respective fields: server name, User name, Password and activate the **Allow Saving password** check box. Click **OK**.
   The ADO connection string is now visible in the connection string field.
6. Click the **Register** button.
   The schema is now registered in Oracle XML Db.

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.



File URL: http://ORCL92:8080/home/SCOTT/purchaseOrders/2001/Apr/A

Open as: ● Auto ○ XML ○ DTD

File load: ● Use cache/proxy ○ Reload

Identification
User: scott Password: ***** □ save

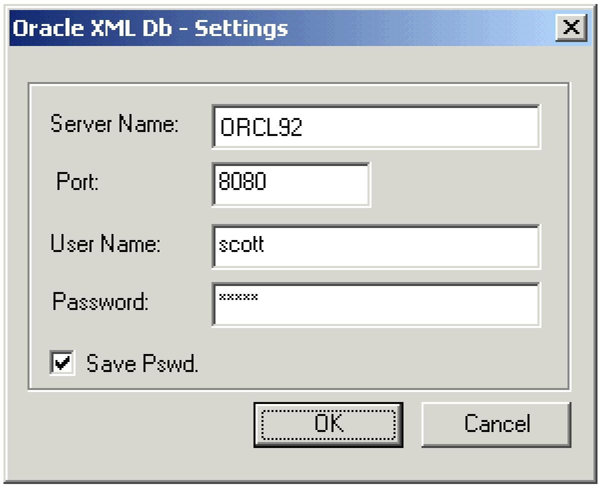
Available files
Server URL: http://ORCL92:8080/

- ORCL92:8080
  - home
    - SCOTT
      - purchaseOrders
        - 1999
        - 2000
        - 2001
          - Apr
            - ADAMS-20011127121051212PST.xml
            - ADAMS-20011127121051483PST.xml

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 **OK** to achieve the same thing. The **New Folder** button adds a new folder, the **Delete** button deletes the currently selected XML file.

Properties

This command displays the Oracle XML Db settings of the currently active query, schema, or XML document in the main window.



Oracle XML Db - Settings

Server Name: ORCL92

Port: 8080

User Name: scott

Password: *****

☑ Save Pswd.

Server Name
The name of the server you are connecting to using the http protocol.

Port
The server port you are using to connect to the server. 8080 is the localhost connection.

User Name
Your user name.

Password
Your password.

Save Pswd.
Retains the password for this XML document.

Adding XML document instances

To add an XML document instance to the database:

1. Open an existing schema saved in the Oracle XML Db using the **Convert | Oracle XML DB | List schemas** command.
2. Select the menu option **DTD/Schema | Generate Sample XML file**.
3. Select the specific options for the XML file, and click **OK**.
   This creates the XML file which is now associated to the schema (please refer to the note below for more details).
4. Select the menu option **File | Save to URL**.
5. Click the **Browse** button and navigate to the folder in which you want to save the XML document instance. Use the New Folder button to create a new folder if necessary.
6. Enter the name of the file after the file path/URL, in the File URL field (please ensure that you are saving the file on the Oracle database server).
7. Click **OK** to save the XML instance file to this location.

Please note:

Since the root node of each document added includes a **noNamespaceschemalocation** attribute that identifies it as an instance of the registered schema, the documents are shredded and stored as objects in the database.

11.10 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.

11.10.1 Text View

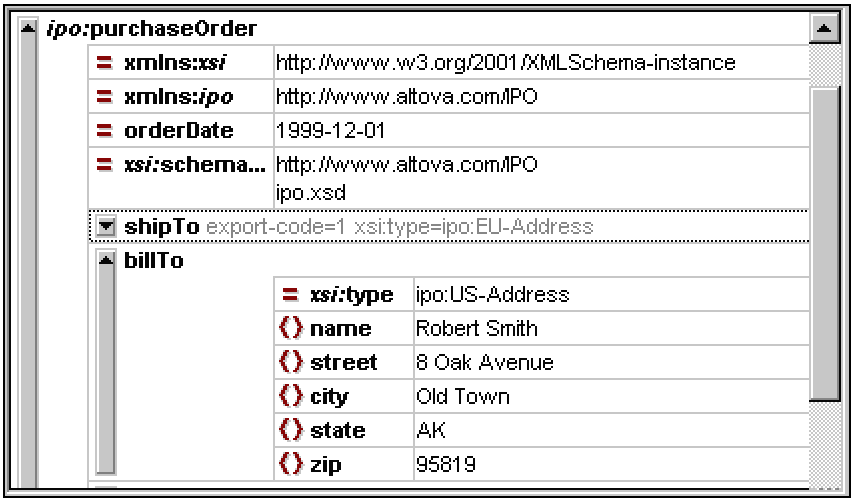
This command **switches** the current view of the document to Text View, which enables you to edit the document in its text form. It supports a number of advanced text editing features, described in detail in Text View section of this document.

Please note: You can configure aspects of the Text View using options available in the various tabs of the Options dialog (**Tools | Options**).

11.10.2 Enhanced Grid View

This command **switches** the current document into Enhanced 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 Enhanced Grid view. This function is available wherever the Enhanced 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.

11.10.3 Schema/WSDL Design View

This command **switches** the current document to Schema/WSDL View. This view is described in detail in the Schema/WSDL Design view section of this documentation.

11.10.4 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 Stylesheet**s (***.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.

11.10.5 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.

11.10.6 Expand

Hotkey: **"+" on the numeric keypad**

This command **expands** the selected element by one level.

The command can be used in the Enhanced Grid view.

In the Enhanced 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.

11.10.7 Collapse

Hotkey: **"-" on the numeric keypad**

This command **collapses** the selected element by one level.

The command can be used in the Enhanced Grid view.

You can expand and collapse any element by clicking on the gray bar to the left of each element.

11.10.8 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 Enhanced Grid view.

11.10.9 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 Enhanced Grid view.

Select the item that you want to work with and choose this command to collapse all other (unselected) elements.

11.10.10Optimal widths

This command **adjusts** the widths of all columns so that the text of the entire document fits into the designated columns.

If you expand and collapse several elements, select the **Optimal widths** command, as only visible items are taken into account when calculating the optimum column widths.

11.10.11Word Wrap

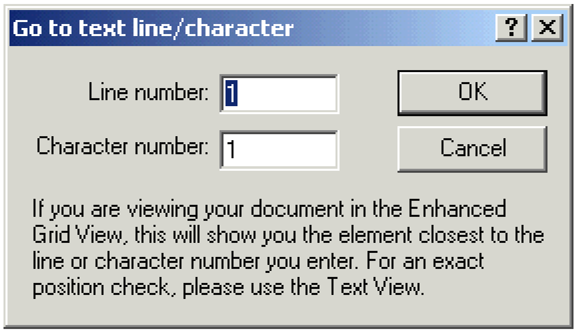
This command enables or disables word wrapping in the **Text view**.

11.10.12Go to line/char...

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:



Line number: 1

Character number: 1

If you are viewing your document in the Enhanced Grid View, this will show you the element closest to the line or character number you enter. For an exact position check, please use the Text View.

This command works in both the Text View and Enhanced 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.

11.10.13Go 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 Enhanced Grid View
- Some characters from within any item in the Text or Enhanced 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.

11.10.14Line Numbers Margin

This command **switches** the line numbering margin on or off in the Text view.

11.10.15Bookmarks Margin

This command **toggles** the Bookmarks Margin on and off in Text View. If a bookmark is present in the Bookmarks Margin and the Bookmarks Margin is then toggled off, the bookmarked line is highlighted. When the Bookmarks Margin is toggled on again, the highlighting is removed and the bookmark appears in the Bookmarks Margin.

11.10.16Source Folding Margin

This command **switches** the source folding margin on or off in the Text view. Clicking the plus or minus symbols in this margin allows you to expand or collapse the respective section of code.

Please note:
You can expand or collapse child "nodes" in the text view:

SHIFT+Click on a plus symbol **collapses** all child nodes of that current node, while

CTRL+Click on a previously collapsed node, **expands** all the child nodes.

11.10.17Indentation Guides

This command switches the indentation guides on or off in the Text view. The indentation guides are the vertical lines, aligned along the whitespace borders, which give you a visual indication of the hierarchical levels.

11.11 Browser Menu

The commands on the **Browser** menu are only available in Browser View.

11.11.1 Back

Backspace

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.

11.11.2 Forward

The **Forward** command is only available once you have used the **Back** command. It moves you forward through previously viewed pages.

11.11.3 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.

11.11.4 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.

11.11.5 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.

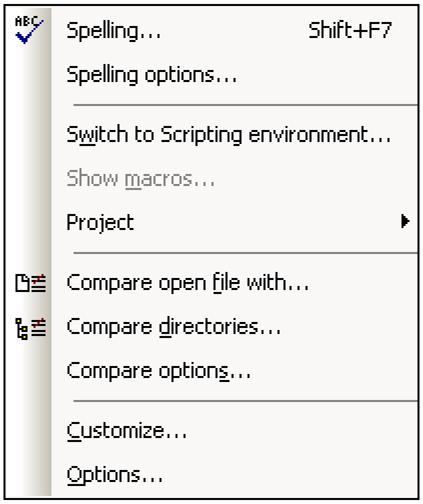
11.11.6 Separate Window

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.

11.12 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
- Assign (or deassign) scripts to a project
- Compare any two files to check for differences
- Compare any two folders to check for differences
- Customize your version of XMLSpy: define your own toolbars, keyboard shortcuts, menus, and macros
- Define the global program settings



11.12.1 Spelling...

XMLSpy includes a built-in spelling checker.

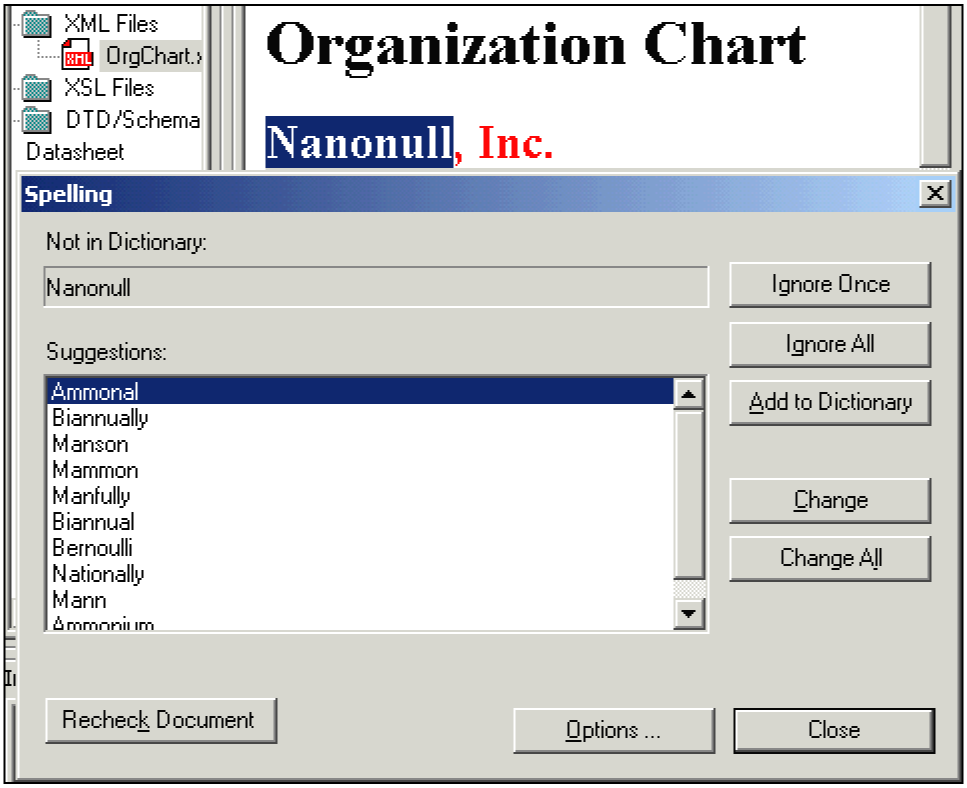
The spelling checker can be used in the following views: Enhanced grid, Text and Authentic View. Please note that you can check any type of XML file. The view you select influences the spelling checker options that are made available. A schema file (*.xsd) can be checked in the Text and Enhanced grid view, but not in the Schema/WSDL Design view.

Spell checking in the **Text** and **Enhanced grid** view presents you with the most options. You can additionally check the spelling of:

- Element content
- Attribute content
- CDATA content
- Comment text

You can also define which specific element or attribute content to check or ignore. These selections are defined in the "Spelling context" dialog box.

The **Tools | Spelling...** command (Shift+F7) opens the Spelling dialog box, and automatically starts checking the currently active XML document (in this screenshot, the `OrgChart.xml` document in Authentic View).



Not in Dictionary:

Nanonull

Suggestions:

Ammonal
Biannually
Manson
Mammon
Manfully
Biannual
Bernoulli
Nationally
Mann
Ammonium

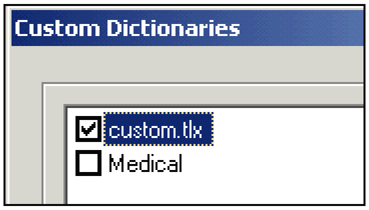
Not in Dictionary
This text box contains the word that cannot be found in any of the existing dictionaries (default and custom dictionaries). You can edit the text here and click **Change** to change it in the XML document. The word is also highlighted in the XML document. The command buttons become active at this point and allow you to decide which action to take.

Suggestions
This list box displays a list of words that resemble the unknown word (supplied from all dictionaries). Double-clicking a word in this list automatically inserts it in the document and continues the spell checking 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 currently active **custom dictionary**. The active dictionary is the dictionary that is selected in the Custom Dictionaries dialog box. To open this dialog, click **Options**, then **Spelling options**, then **Custom Dictionaries**.



☑ custom.tlx
□ Medical

Change
This command replaces the currently highlighted word in the XML document with the selected word in the Suggestions list box.

Change all
This command replaces all occurrences of the currently highlighted word in the XML document, with the selected word in the Suggestions list box.

Recheck
The Recheck button restarts the check from the beginning of the document.

Close
This command closes the Spelling dialog box.

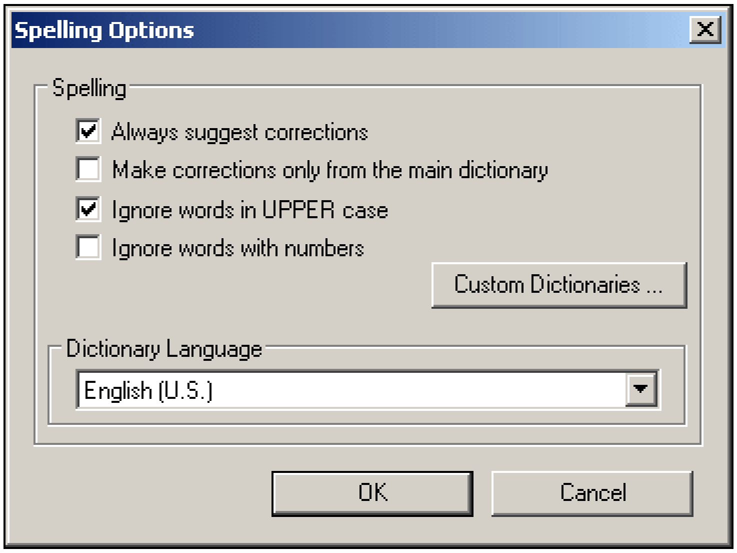
Options...
This command opens a dialog box depending on the current view.

- If opened from Authentic View, the Options dialog box is opened.
- If opened from the Text or Enhanced Grid view, the Spelling context dialog box is opened.

11.12.2 Spelling options...

The **Tools | Spelling options** command opens either the **Spelling Options** or **Spelling context** dialog box, depending on the view you are using to display your XML document.

Spelling Options dialog (Browser View and Schema/WSDL View)
When viewing an XML document in Browser View or Schema/WSDL View, this command opens the Spelling Options dialog box. Use this dialog box to define the global spelling checker options.



Spelling Options

Spelling

☑ Always suggest corrections
□ Make corrections only from the main dictionary
☑ Ignore words in UPPER case
□ Ignore words with numbers

Dictionary Language: English (U.S.)

Always suggest corrections:
Activating this option causes suggestions (from all of the dictionaries) 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 default dictionary to be used; none of the custom dictionaries are scanned for suggestions. It also disables the **Custom Dictionaries** button,

preventing any editing of the custom dictionaries.

Ignore words in UPPER case:
Activating this option causes all upper case words to be ignored.

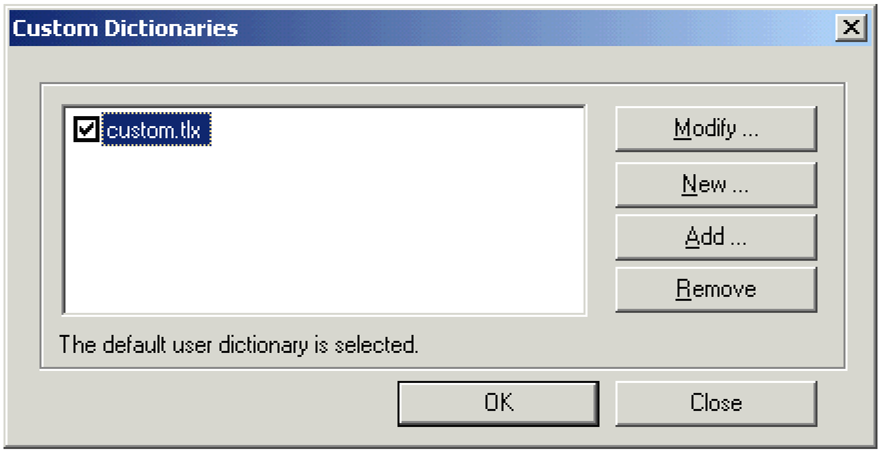
Ignore words with numbers:
Activating this number causes all words containing numbers to be ignored.

Dictionary Language
Use this combo box to select the dictionary language for the spelling checking. The default installation allows you to select English. Other language dictionaries will be made available on the Altova download web site.

Custom dictionaries
The **Custom Dictionaries...** button allows you to:

- modify an existing dictionary (add or delete dictionary entries)
- create a totally new dictionary
- add an existing dictionary
- remove an existing dictionary

When you start the spell checking process, all dictionaries listed in the Custom Dictionaries list box are searched. If you want to limit the search to specific dictionaries, use the **Remove** button to delete those you do not want searched.

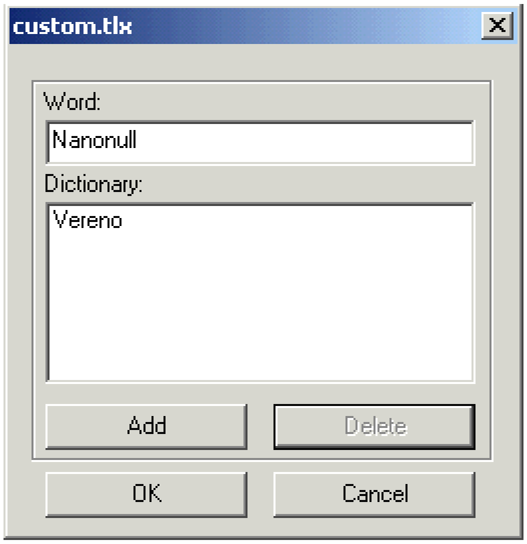


☑ custom.tlx

The default user dictionary is selected.

To add a new dictionary entry:

1. In the Custom Dictionaries dialog, click the name of the custom dictionary to which you want to add an entry, and click **Modify...**
   This opens the dictionary highlighted in the list box (`custom.tlx` in this case). A prompt appears if none of the dictionaries has been selected.



Word: Nanonull

Dictionary: Vereno

2. Click in the **Word:** field and enter the new dictionary entry (this automatically activates the **Add** button).
3. Click **Add** to add it to the dictionary.
4. Click **OK**.

To delete an entry from the dictionary:

1. In the Custom Dictionaries dialog, click the name of the custom dictionary from which you want to delete an entry, and click **Modify...**
   This opens the dictionary highlighted in the list box (`custom.tlx` in this case). A prompt appears if none of the dictionaries has been selected.
2. Click the word in the Dictionary list box to highlight it, and click **Delete**.
3. Click **OK**.

To add a new dictionary:

1. In the Custom Dictionaries dialog, click **New**, and enter the name of the new custom dictionary in the **File name** field.
2. Click **Save** to save the dictionary.
3. You can now add entries to the dictionary using the procedure above, or the **Add to Dictionary** button while performing a spell check.

To add an existing dictionary:

Use this option to add previously removed (or third party) dictionaries.

1. In the Custom Dictionaries dialog, click **Add** and enter the name of the dictionary in the **File name** field.
   Dictionaries have a *.**tlx** extension.
2. Click **Open** to add the dictionary to the Custom dictionary list.

Please note:

It is not mandatory for a dictionary to have a `*.tlx` extension. It can have any extension, or no extension at all, and still be added to the dictionary list box.

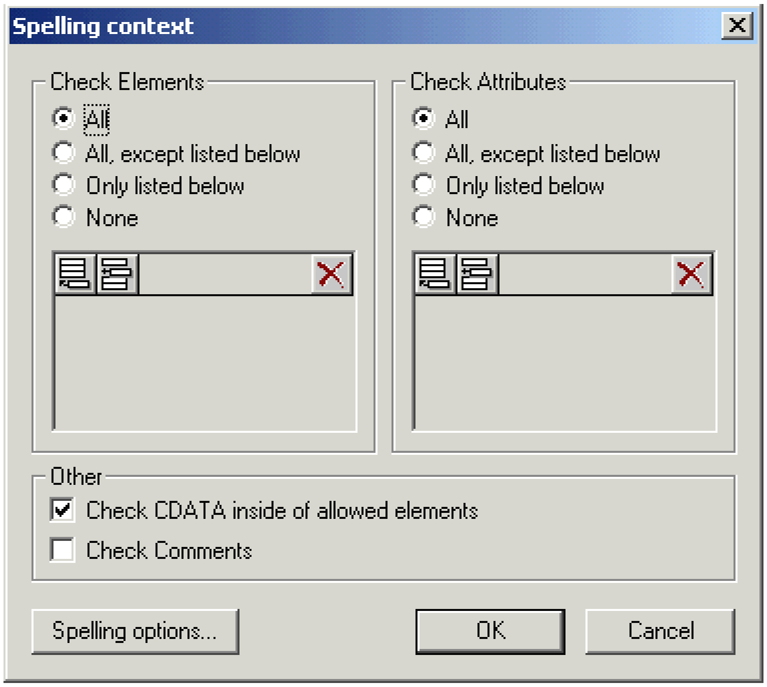
To remove a dictionary:

- In the Custom Dictionaries dialog, click the dictionary name in the list and click **Remove** . This removes the dictionary from the list, but does not physically delete it from your hard disk.

Spelling context dialog- **Text** and **Enhanced Grid** view

When viewing an XML document in the Text or Enhanced Grid view, this command opens the "Spelling context" dialog box. Starting the spelling checking in the **Text** and **Enhanced grid** view presents you with the most options, you can define the specific content which is to be checked:

- Element content
- Attribute content
- CDATA content
- Comment text



Spelling context

Check Elements
● All
○ All, except listed below
○ Only listed below
○ None

Check Attributes
● All
○ All, except listed below
○ Only listed below
○ None

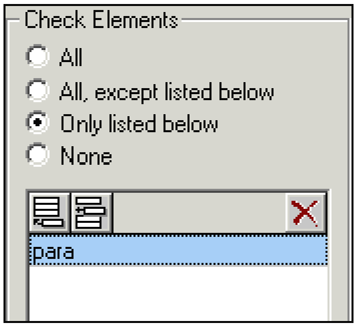
Other
☑ Check CDATA inside of allowed elements
□ Check Comments

The **Spelling options...** button opens the Spelling Options dialog box, which allows you to define the global spelling checker options.

To define specific element or attribute "content" you want to check:

1. Click the **All, except listed below** or **Only listed below** radio button.

2. Click the Append element/attribute icon ▤, and enter the tag name containing the text you want checked (e.g., para). Press **Enter** to confirm.



Check Elements
○ All
○ All, except listed below
● Only listed below
○ None

para

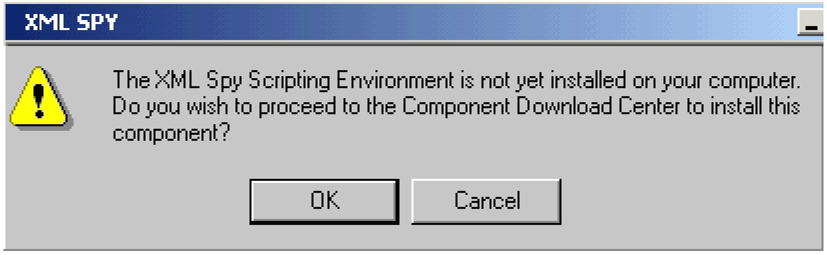
3. Click **OK** to confirm these settings, and click it again in the Spelling dialog box to start the spelling checker with these new settings.
The only **element content** to be checked will be the text between the **para** tags, attribute content will also be checked.

To delete a tag name:

- Select the tag name, and press the **Delete** or **Backspace** key.

11.12.3 Switch to scripting environment

The scripting environment is currently available as a downloadable component. If you have not downloaded and installed the scripting component, you will be prompted to do so when you select the **Switch to Scripting environment** menu item.



The XML Spy Scripting Environment is not yet installed on your computer. Do you wish to proceed to the Component Download Center to install this component?

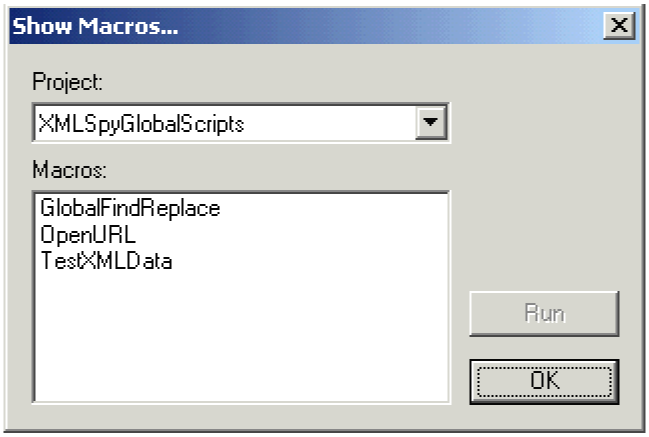
Click **OK** to connect with the Altova Component Download Center from where you can choose to download the "Scripting Environment".

After you have downloaded the component, double-click on **Spyscripting.exe** to install it.

This command switches to the scripting environment, a predefined global scripting project is active the first time you start the form editor. Please see the Scripting section in this manual for more information.

11.12.4 Show macros

This command opens a dialog box which displays a list of all macros defined in the global scripting project and in the scripting project associated with the current project opened in XMLSpy.



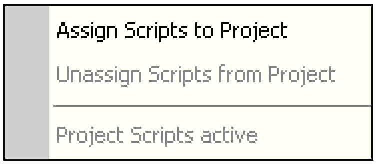
Project: XMLSpyGlobalScripts

Macros:
GlobalFindReplace
OpenURL
TestXMLData

The **Run** button in the dialog box calls the selected macro.

11.12.5 Project

The **Tools | Options** command opens a further submenu from where you can assign (or de-assign) scripts to a project.



Please note:
You have to have a XMLSpy project open for these menu items to become active.

Assign script to Project

This command enables you to assign scripts to a scripting project.

Unassign Scripts from Project

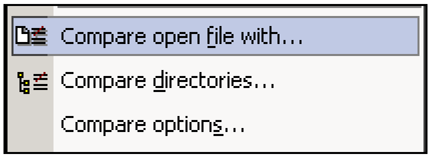
This command enables you to de-assign scripts from a scripting project.

Project Scripts active

This command allows you to activate or temporarily deactivate project scripts.

11.12.6 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.

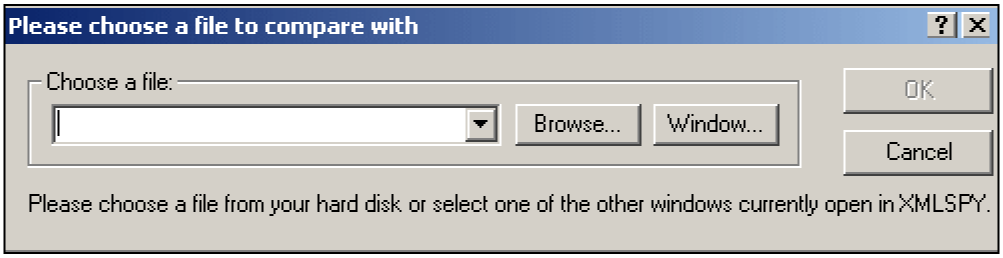
Compare Open File with

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...**. The following dialog

appears:



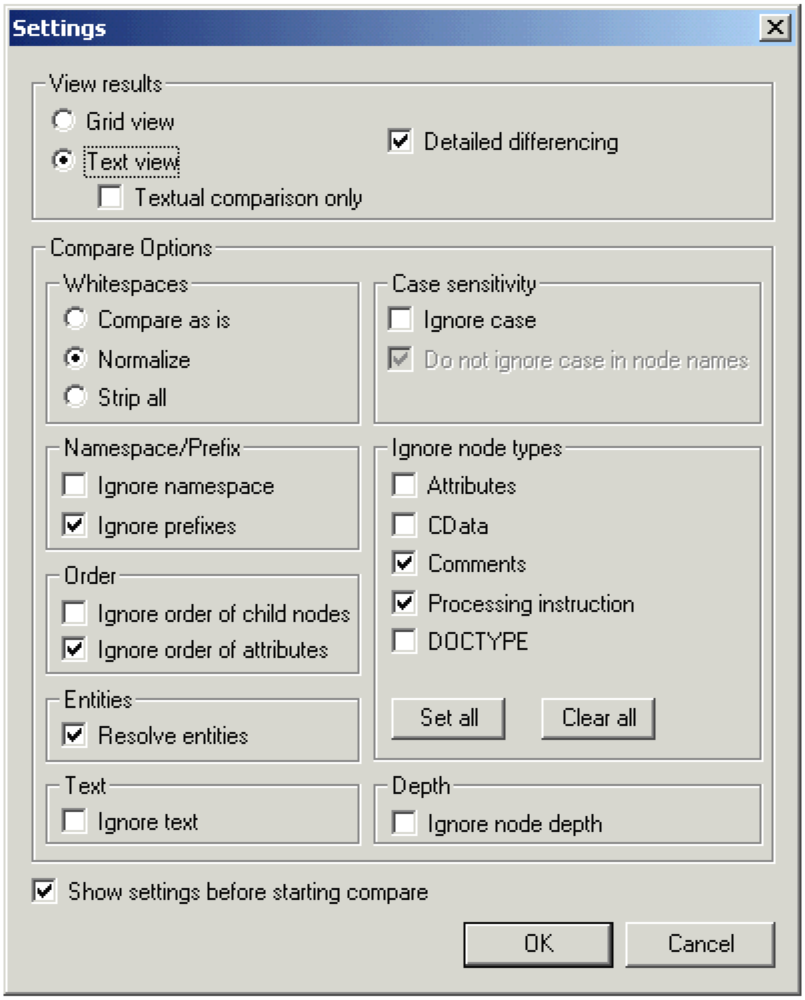
Please choose a file to compare with

Choose a file:

Browse... Window... OK Cancel

Please choose a file from your hard disk or select one of the other windows currently open in XMLSPY.

2. Browse for the file you wish to compare the open file with, and select it. Alternatively, if the file you wish to select is open (but not active) in the Main Window, or if the file is in a project, click the **Window...** button, and select the file.
3. Click **OK**. The second file is opened, and the Settings dialog appears:



Settings

View results
- ○ Grid view
- ● Text view
- □ Textual comparison only
- ☑ Detailed differencing

Compare Options

Whitespaces
- ○ Compare as is
- ● Normalize
- ○ Strip all

Case sensitivity
- □ Ignore case
- ☑ Do not ignore case in node names

Namespace/Prefix
- □ Ignore namespace
- ☑ Ignore prefixes

Ignore node types
- □ Attributes
- □ CData
- ☑ Comments
- ☑ Processing instruction
- □ DOCTYPE

Set all Clear all

Order
- □ Ignore order of child nodes
- ☑ Ignore order of attributes

Entities
- ☑ Resolve entities

Text
- □ Ignore text

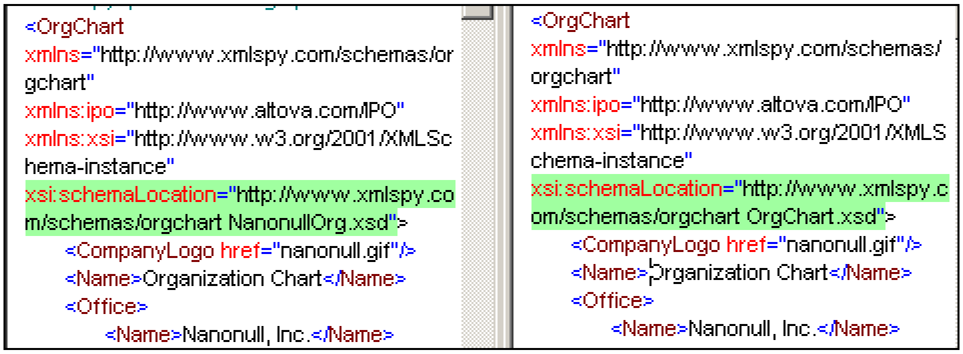
Depth
- □ Ignore node depth

☑ Show settings before starting compare

OK Cancel

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.

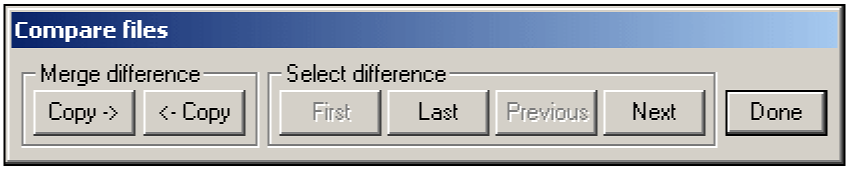
4. Select the required settings, then click **OK**. The two files now appear in the Main Window. Lines that are different are highlighted in green. One difference is selected at a time and is indicated in both files in a darker green color.



```
<OrgChart
xmlns="http://www.xmlspy.com/schemas/or
gchart"
xmlns:ipo="http://www.altova.com/IPO"
xmlns:xsi="http://www.w3.org/2001/XMLSc
hema-instance"
xsi:schemaLocation="http://www.xmlspy.co
m/schemas/orgchart NanonullOrg.xsd">
    <CompanyLogo href="nanonull.gif"/>
    <Name>Organization Chart</Name>
    <Office>
        <Name>Nanonull, Inc.</Name>
```

```
<OrgChart
xmlns="http://www.xmlspy.com/schemas/
orgchart"
xmlns:ipo="http://www.altova.com/IPO"
xmlns:xsi="http://www.w3.org/2001/XMLS
chema-instance"
xsi:schemaLocation="http://www.xmlspy.c
om/schemas/orgchart OrgChart.xsd">
    <CompanyLogo href="nanonull.gif"/>
    <Name>Organization Chart</Name>
    <Office>
        <Name>Nanonull, Inc.</Name>
```

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**.

Please note:

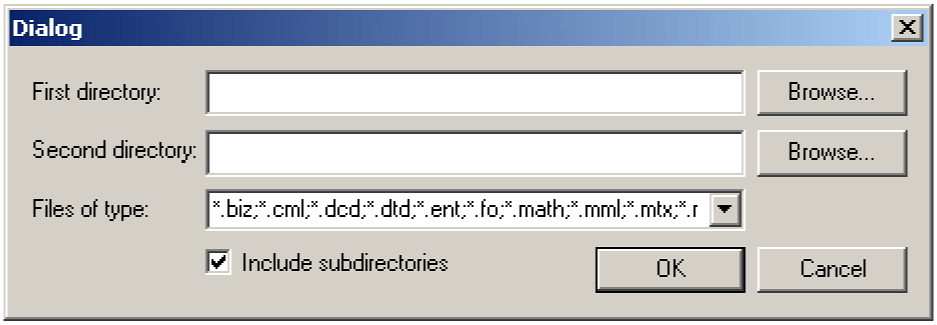
- While the Compare session is active, no editing or change of views is allowed. Any attempt to edit or change the view of either file will pop up a message warning that the Compare session will be ended.
- Compare settings can be changed during a Compare session (by selecting **Tools | Compare options...**), 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.



First directory:

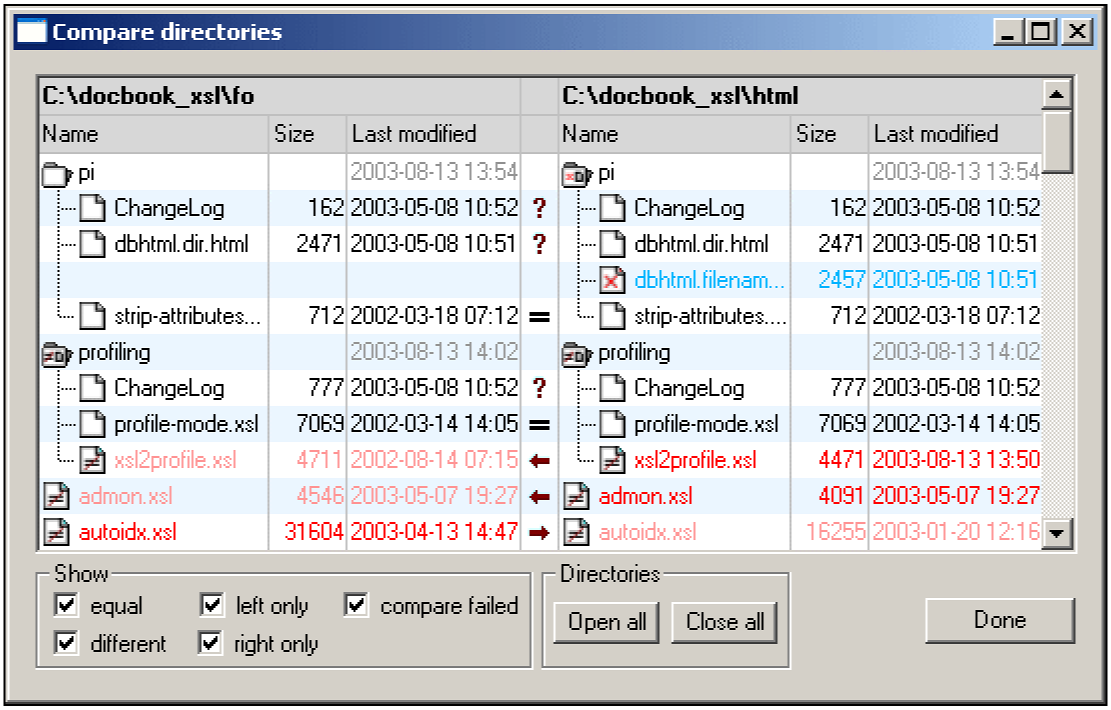
Second directory:

Files of type: *.biz;*.cml;*.dcd;*.dtd;*.ent;*.fo;*.math;*.mml;*.mtx;*.r

☑ Include subdirectories

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.
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:



C:\docbook_xsl\fo				C:\docbook_xsl\html		
Name	Size	Last modified		Name	Size	Last modified
pi		2003-08-13 13:54		pi		2003-08-13 13:54
ChangeLog	162	2003-05-08 10:52	?	ChangeLog	162	2003-05-08 10:52
dbhtml.dir.html	2471	2003-05-08 10:51	?	dbhtml.dir.html	2471	2003-05-08 10:51
				dbhtml.filenam...	2457	2003-05-08 10:51
strip-attributes...	712	2002-03-18 07:12	=	strip-attributes....	712	2002-03-18 07:12
profiling		2003-08-13 14:02		profiling		2003-08-13 14:02
ChangeLog	777	2003-05-08 10:52	?	ChangeLog	777	2003-05-08 10:52
profile-mode.xsl	7069	2002-03-14 14:05	=	profile-mode.xsl	7069	2002-03-14 14:05
xsl2profile.xsl	4711	2002-08-14 07:15	←	xsl2profile.xsl	4471	2003-08-13 13:50
admon.xsl	4546	2003-05-07 19:27	←	admon.xsl	4091	2003-05-07 19:27
autoidx.xsl	31604	2003-04-13 14:47	→	autoidx.xsl	16255	2003-01-20 12:16

Show
☑ equal ☑ left only ☑ compare failed
☑ different ☑ right only

Directory symbols
All directory names are given in black.

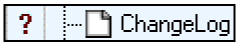
This directory contains files, all of which either cannot be compared or are not different from the corresponding file in the compared directory.

This directory contains one or more files that do not exist in the compared directory.

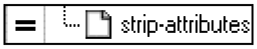
This directory contains one or more files that are different from the corresponding file in the compared directory.

Please note: If a directory contains both a file that does not exist in the compared directory as well as a file that is different from the corresponding file in the compared directory, then the directory will be shown with the icon .

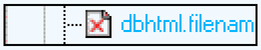
File symbols
The colors in which file names appear depend on their compare status. These colors are noted below.



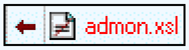
This file cannot be compared (with the corresponding file in the compared directory). A question mark appears in the middle column. The file name appears in black.



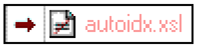
This file is not different from the corresponding file in the compared directory. An equals-to sign appears in the middle column. The file name appears in black.



This file does not exist in the compared directory. The middle column is empty. The file name appears in blue.



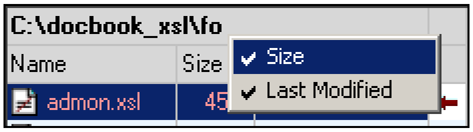
This file is different from the corresponding file in the compared directory, and the last modification to this file is more recent than the last modification to the corresponding file. An arrow pointing towards the older file appears in the middle column. The file name appears in dark red.



This file is different from the corresponding file in the compared directory, and the last modification to this file is older than the last modification to the corresponding file in the compared directory. An arrow pointing towards this (the older) file appears in the middle column. The file name appears in light red.

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.

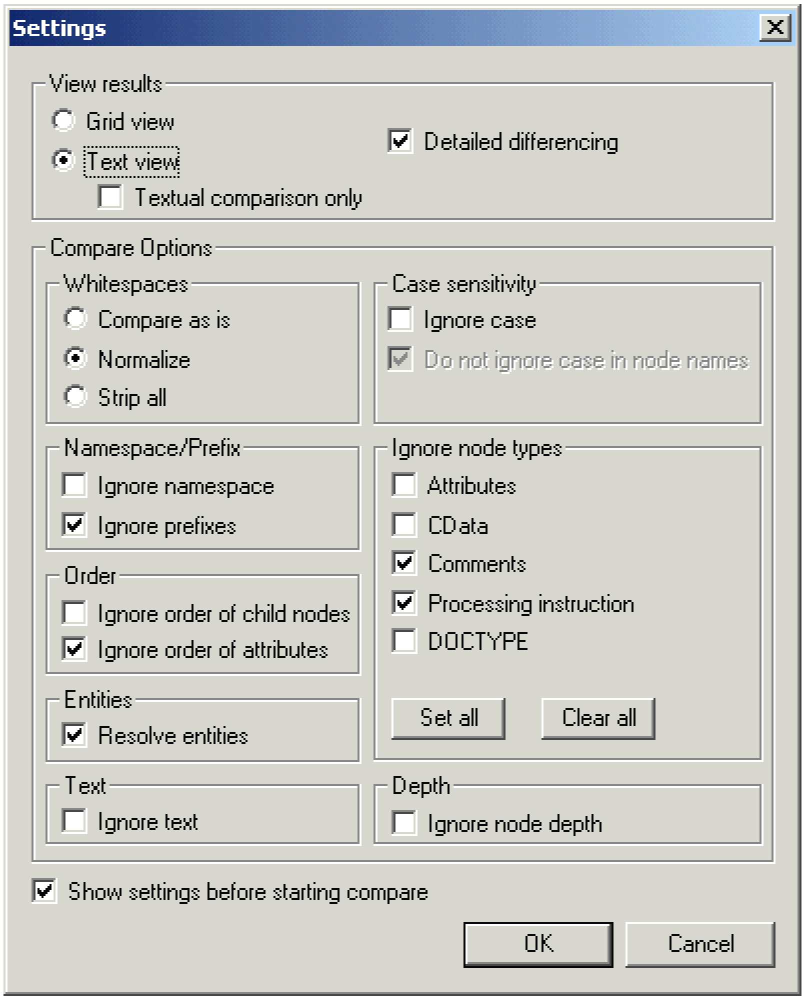


- Change column widths by dragging columns.
- The Result window can be maximized, minimized, and resized.

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.



Settings

View results

○ Grid view

● Text view

□ Textual comparison only

☑ Detailed differencing

Compare Options

Whitespaces

○ Compare as is

● Normalize

○ Strip all

Case sensitivity

□ Ignore case

☑ Do not ignore case in node names

Namespace/Prefix

□ Ignore namespace

☑ Ignore prefixes

Ignore node types

□ Attributes

□ CData

☑ Comments

☑ Processing instruction

□ DOCTYPE

Order

□ Ignore order of child nodes

☑ Ignore order of attributes

Entities

☑ Resolve entities

Text

□ Ignore text

Depth

□ Ignore node depth

☑ Show settings before starting compare

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, and DOCTYPE statements.

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.

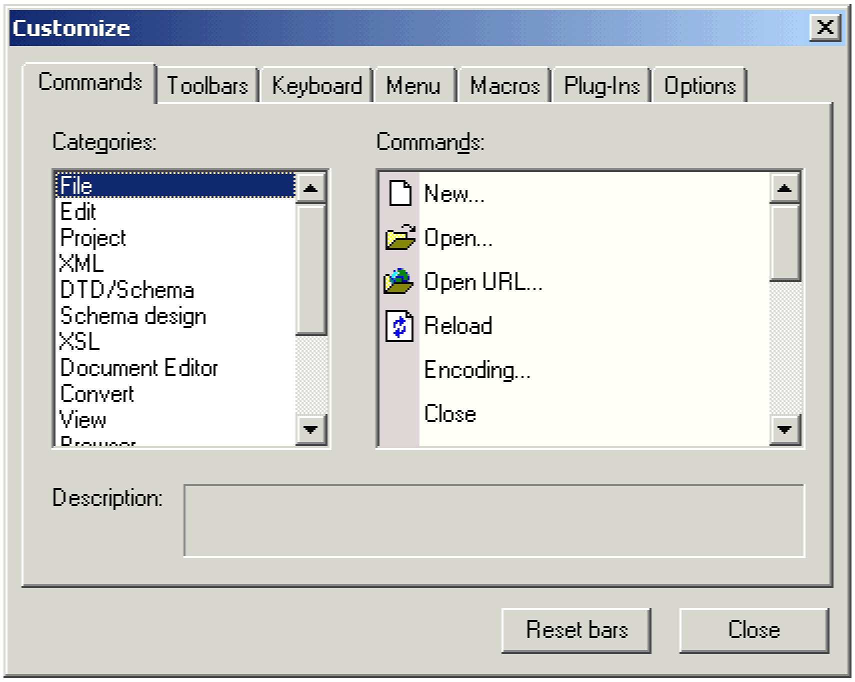
If this command is unchecked, then the Compare session will start directly when a comparison is invoked.

11.12.7 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 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 you 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.

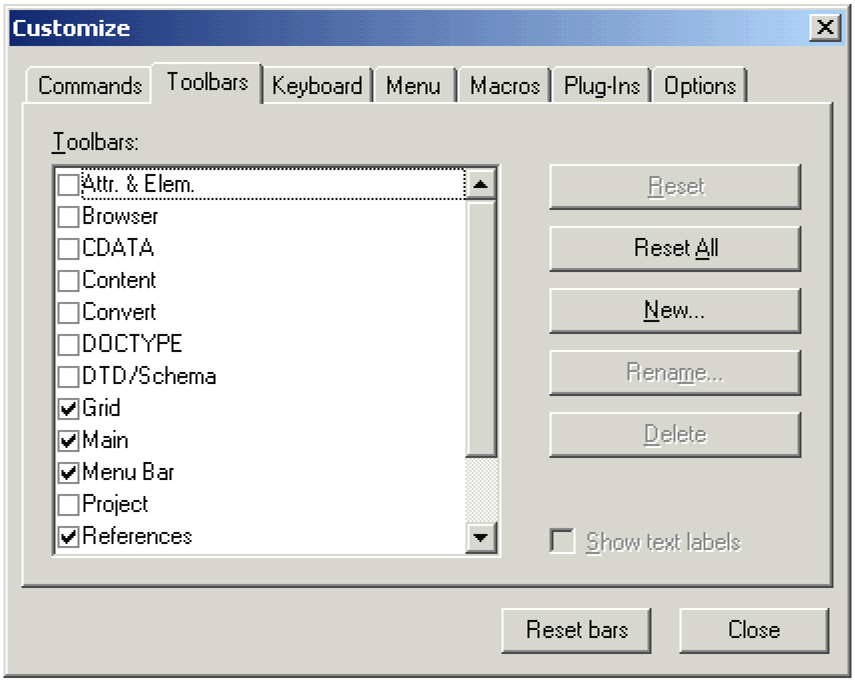
Toolbars

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 the Grid, Schema/WSDL design and Text view are valid in those views. The Browser view toolbars are independent of all the other views.



Toolbars:
- □ Attr. & Elem.
- □ Browser
- □ CDATA
- □ Content
- □ Convert
- □ DOCTYPE
- □ DTD/Schema
- ☑ Grid
- ☑ Main
- ☑ Menu Bar
- □ Project
- ☑ References

□ Show text labels

To activate or deactivate a toolbar:

1. Click the check box to activate (or deactivate) the specific toolbar.

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.

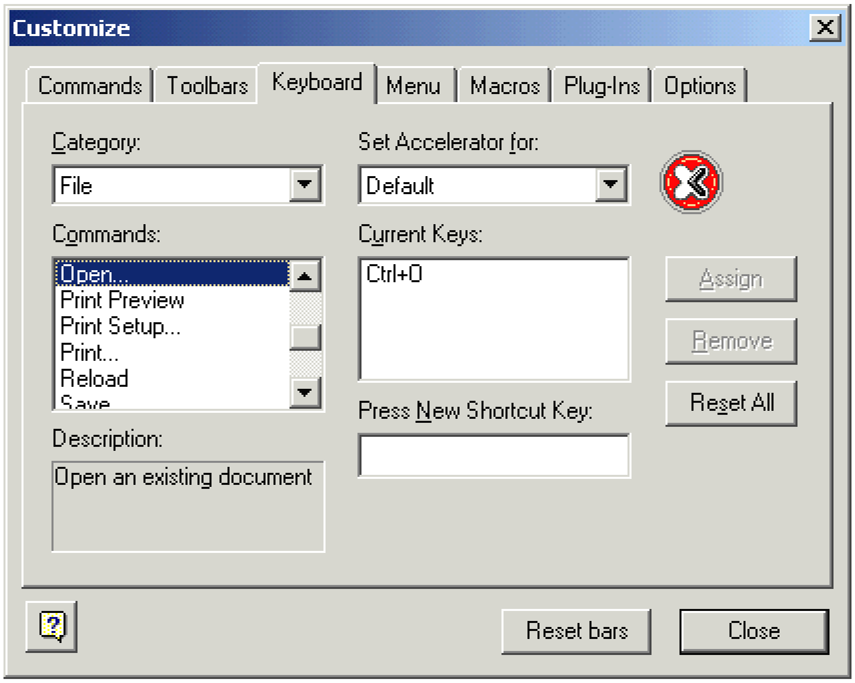
To delete a toolbar:

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.



Customize

Category: File

Set Accelerator for: Default

Commands: Open... Print Preview Print Setup... Print... Reload Save

Current Keys: Ctrl+O

Press New Shortcut Key:

Description: Open an existing document

To assign a new Shortcut to a command:

1. Select the All Commands category using the **Category** combo box.
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/Enhanced Grid view)
CTRL + F	Find
F3	Find Next

CTRL + H	Replace
CTRL + I	Append Attribute
CTRL + E	Append Element
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:

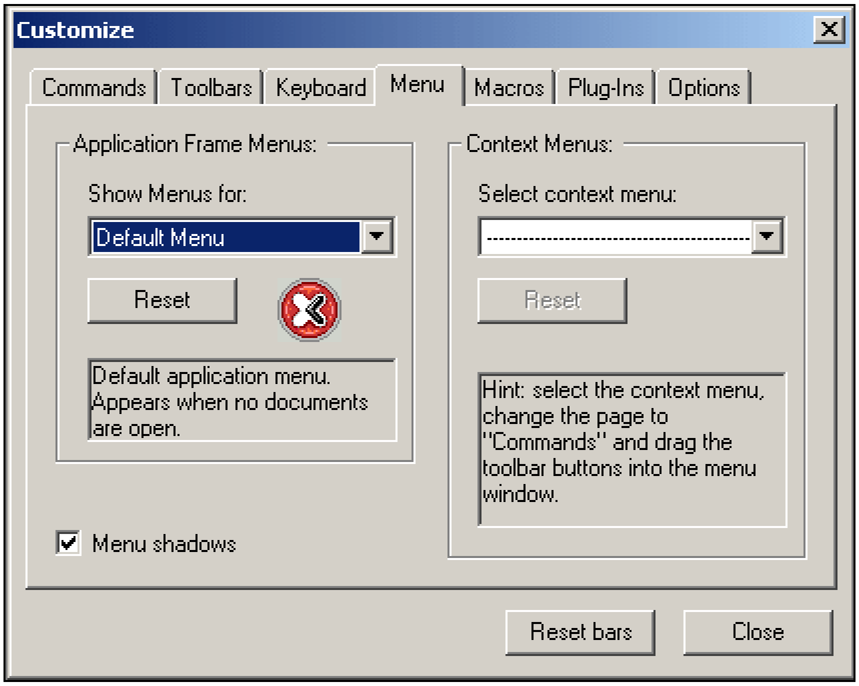
Hotkeys by function

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
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
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

Menu

The **Menu** tab allows you to customize the main menu bars as well as the (popup - right click) context menus.



Show Menus for: Default Menu

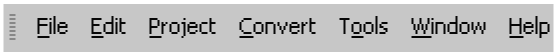
Default application menu. Appears when no documents are open.

Hint: select the context menu, change the page to "Commands" and drag the toolbar buttons into the menu window.

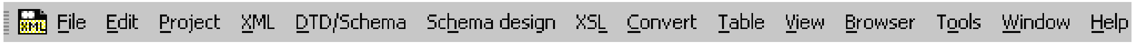
☑ Menu shadows

You can customize both the Default and XMLSpy menu bars.

The **Default** menu 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.

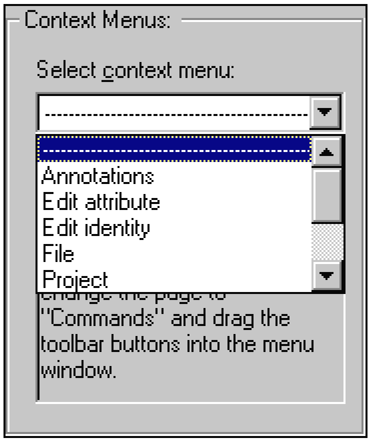
2. Drag the command away from the menu, and drop it as soon as the check mark icon appears below the mouse pointer.

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 animations (only prior to Windows 2000)

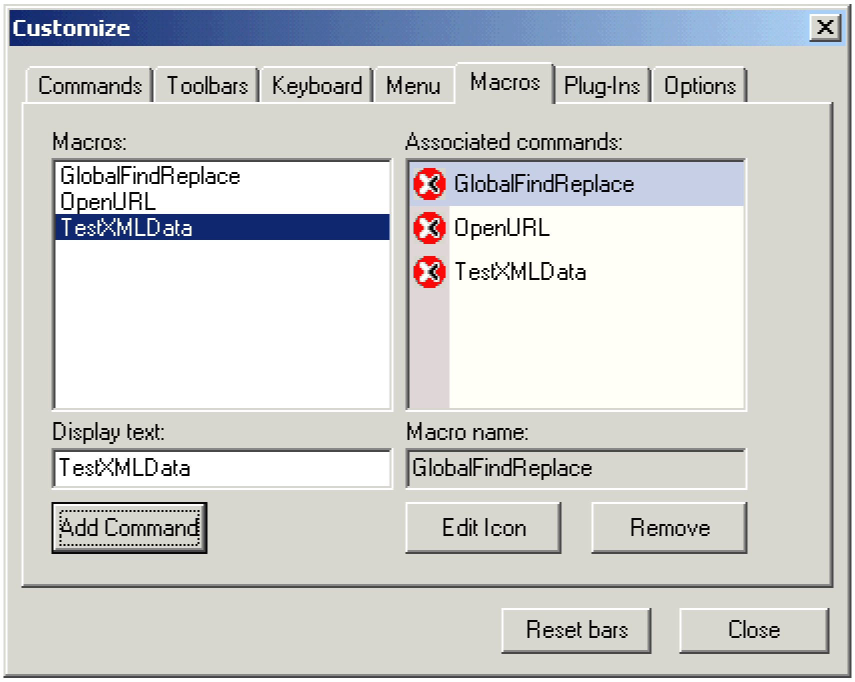
- Select one of the menu animations from the combo box, if you want animated menus. Please note that the combo box is only visible in Windows versions prior to Windows 2000. For Windows 2000 and later, these settings have to be changed in the Effects tab of the Display properties dialog box. Double-click the Display icon in the Control Panel to open the 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.



Macros:
GlobalFindReplace
OpenURL
TestXMLData

Associated commands:
GlobalFindReplace
OpenURL
TestXMLData

Display text: TestXMLData

Macro name: GlobalFindReplace

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:

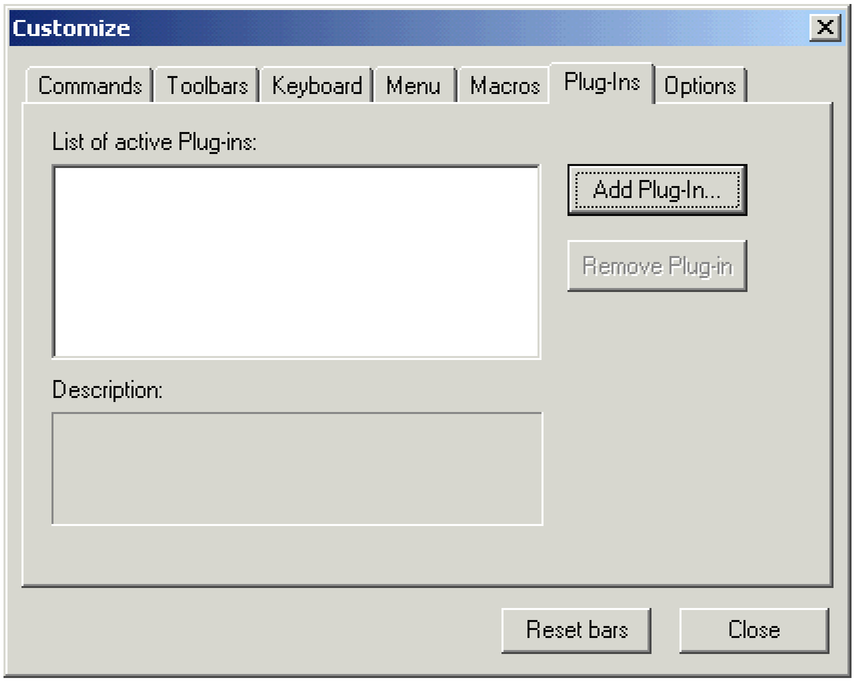
- Click the **Remove** button.

Plug-Ins

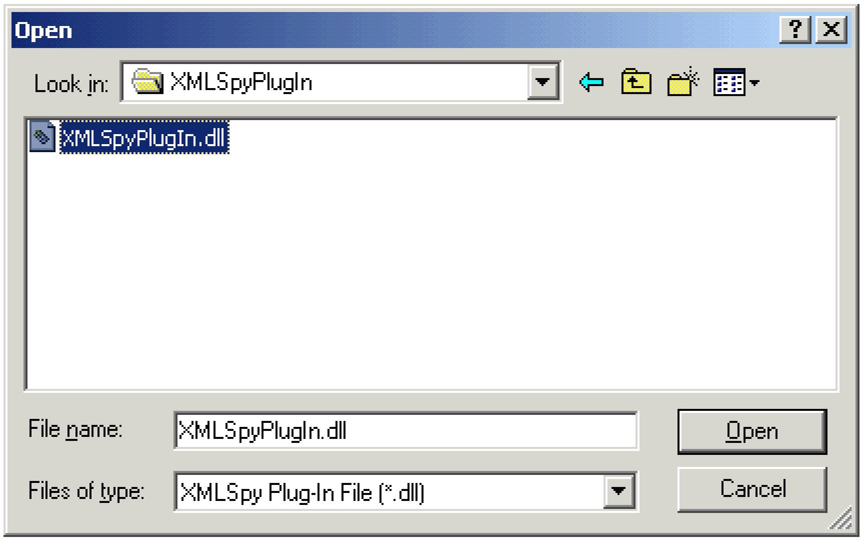
The Plug-Ins tab allows you to place plug-ins in a toolbar or menu.

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).



File name: XMLSpyPlugIn.dll

Files of type: XMLSpy Plug-In File (*.dll)

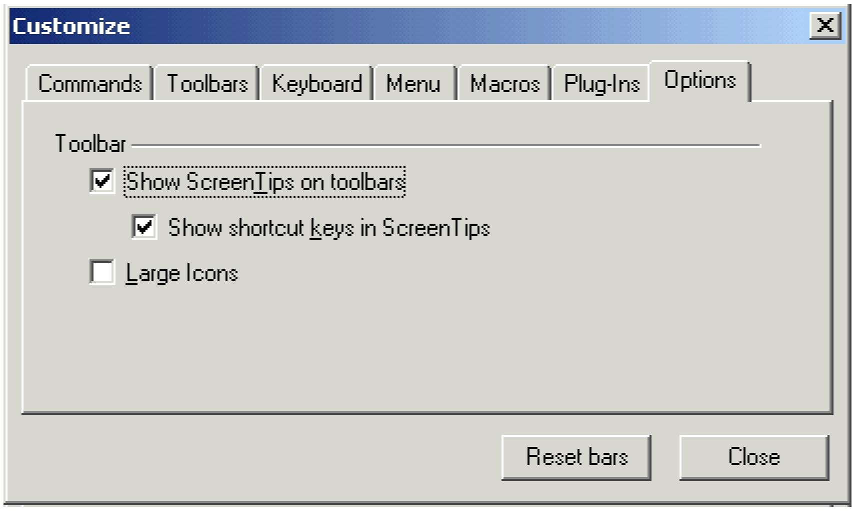
3. Click the **Open** button to install the plug-in.
   The plug-in name appears in the "list of active plug-ins" list, and the plug-in icon(s) appears in a new toolbar.

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.

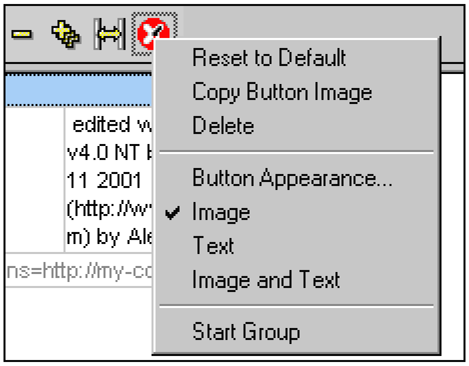
When active, the **Large icons** check box switches between the standard size icons, and larger versions of the icons.

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.

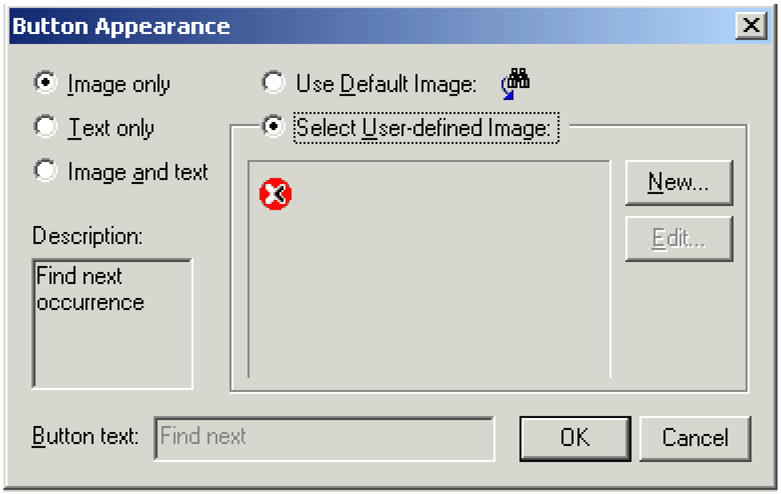
Delete
This option deletes the icon or menu you right click. Deleting a menu deletes all menu options contained in it!

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.



● Image only
○ Text only
○ Image and text
○ Use Default Image:
● Select User-defined Image:

Description: Find next occurrence

Button text: Find next

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.

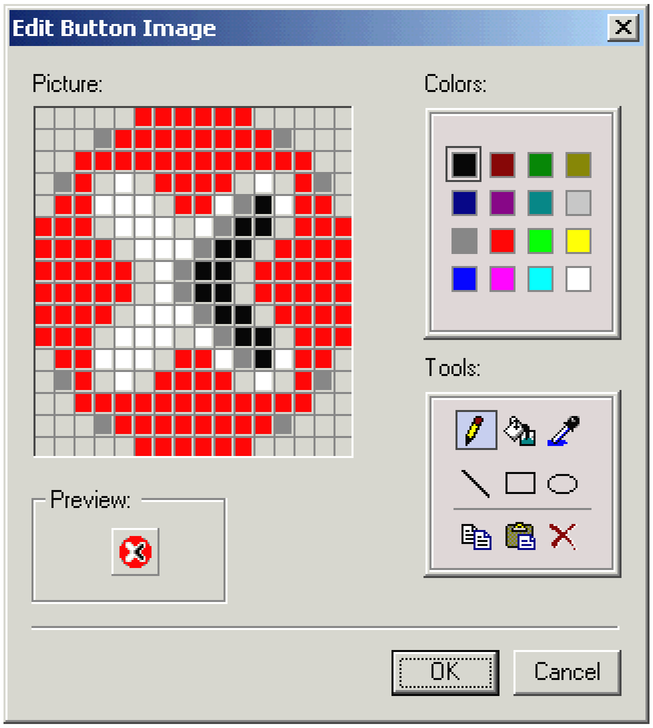


Image
This option changes the graphical display of an icon to the text representing it.

Text
This option changes the textual description of a function to the graphical image (icon) representing that function.

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.

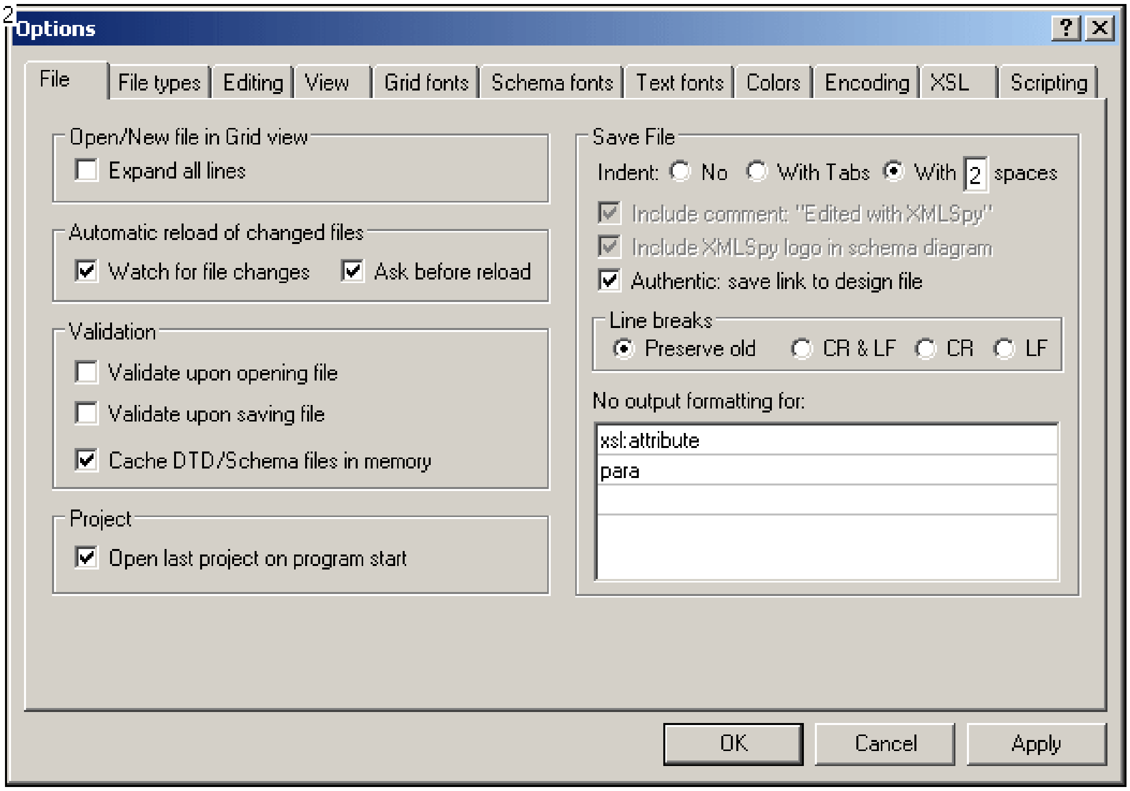
11.12.8 Options

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
□ Expand all lines

Automatic reload of changed files
☑ Watch for file changes ☑ Ask before reload

Validation
□ Validate upon opening file
□ Validate upon saving file
☑ Cache DTD/Schema files in memory

Project
☑ Open last project on program start

Save File
Indent: ○ No ○ With Tabs ● With 2 spaces
☑ Include comment: "Edited with XMLSpy"
☑ Include XMLSpy logo in schema diagram
☑ Authentic: save link to design file

Line breaks
● Preserve old ○ CR & LF ○ CR ○ LF

No output formatting for:
xsl:attribute
para

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. 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 you select **Edit | Pretty-Print XML Text**, indentation is made in your XML file as specified in this pane, i.e., with tab characters (`#x09`), the number of spaces specified, or without any indentation. 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.

When saving an XML document, XMLSpy includes a short comment `<!-- Edited with`

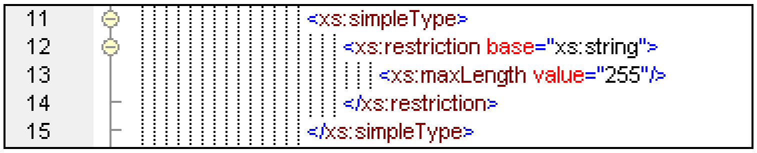
`XMLSpy http://www.altova.com -->` near the top of the file. This option can only be deactivated by licensed users, and takes effect when editing or saving files in the Enhanced Grid or Schema Design View.

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.

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).

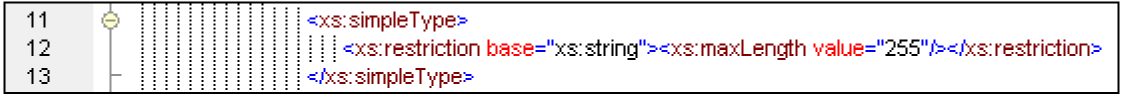
No output formatting for
In Text View, the indentation of an element can be made to reflect its position in the element hierarchy (*see* **Save File**). You can, however, override this indentation for individual elements. To do this, enter the element name in the **No output formatting for** field. All 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:



```
<xs:simpleType>
	<xs:restriction base="xs:string">
		<xs:maxLength value="255"/>
	</xs:restriction>
</xs:simpleType>
```

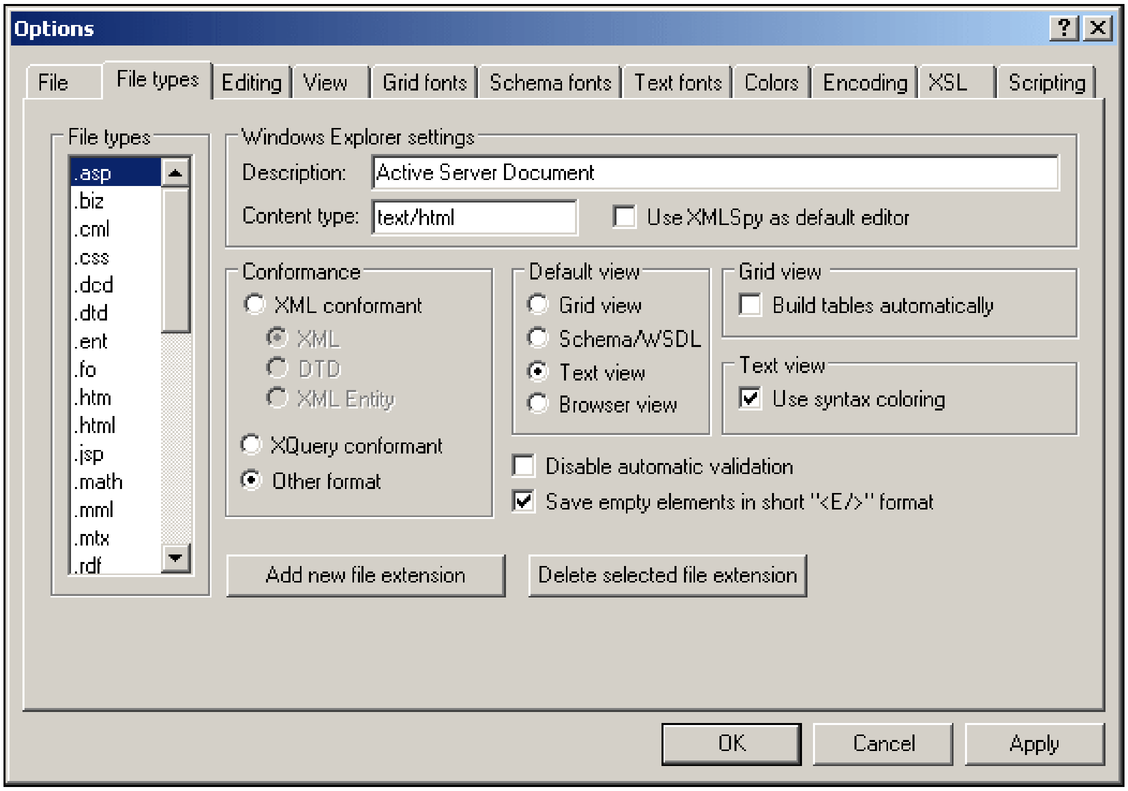
No output formatting has been specified for element `xs:restriction`:



```
<xs:simpleType>
	<xs:restriction base="xs:string"><xs:maxLength value="255"/></xs:restriction>
</xs:simpleType>
```

File types

The **File types** tab allows you to customize the behavior of XMLSpy on a per-file-type basis.



File types
.asp
.biz
.cml
.css
.dcd
.dtd
.ent
.fo
.htm
.html
.jsp
.math
.mml
.mtx
.rdf

Windows Explorer settings

Description: Active Server Document

Content type: text/html

☐ Use XMLSpy as default editor

Conformance

○ XML conformant

● XML

○ DTD

○ XML Entity

○ XQuery conformant

● Other format

Default view

○ Grid view

○ Schema/WSDL

● Text view

○ Browser view

Grid view

☐ Build tables automatically

Text view

☑ Use syntax coloring

☐ Disable automatic validation

☑ Save empty elements in short "<E/>" format

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, 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 Enhanced Grid View should automatically build tables.

Text View
This text 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.

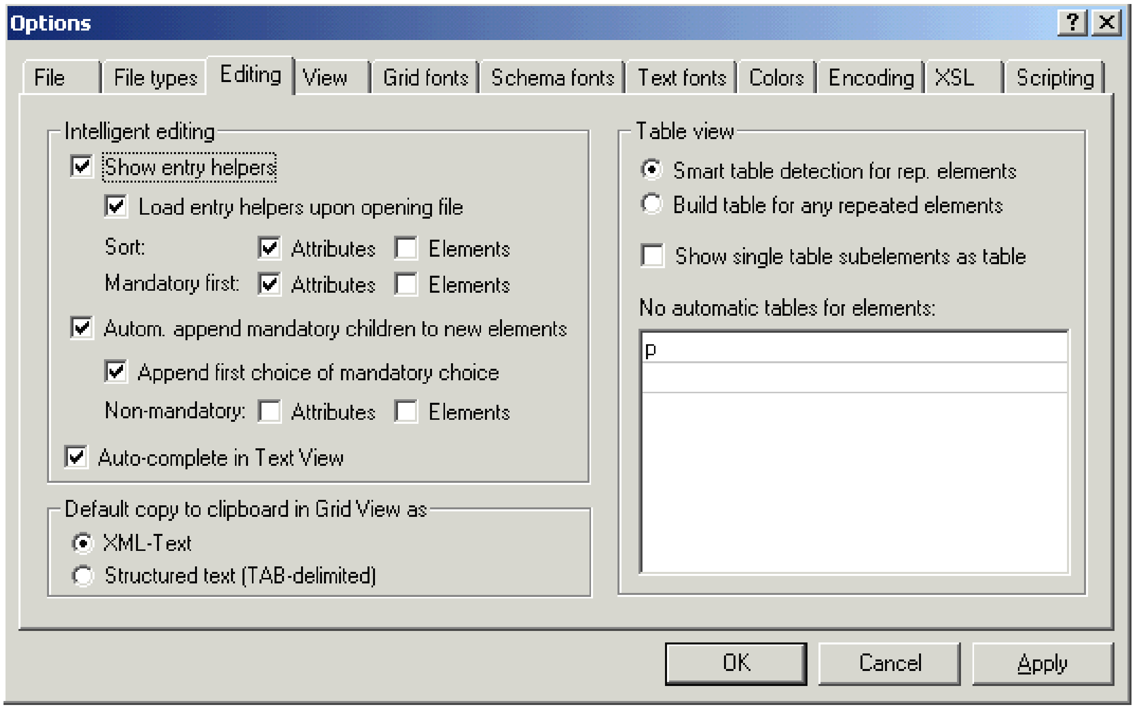
Save empty elements in short <E/> format
Some applications that use XML documents or output generated from XML documents may have problems understanding the short `<Element/>` form for empty elements defined in the XML 1.0 Specification. You can instruct XMLSpy to save elements in the longer (but also 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.

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
☑ Append first choice of mandatory choice
Non-mandatory: ☐ Attributes ☐ Elements
☑ Auto-complete in Text View

Default copy to clipboard in Grid View as
● XML-Text
○ Structured text (TAB-delimited)

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

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.

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.

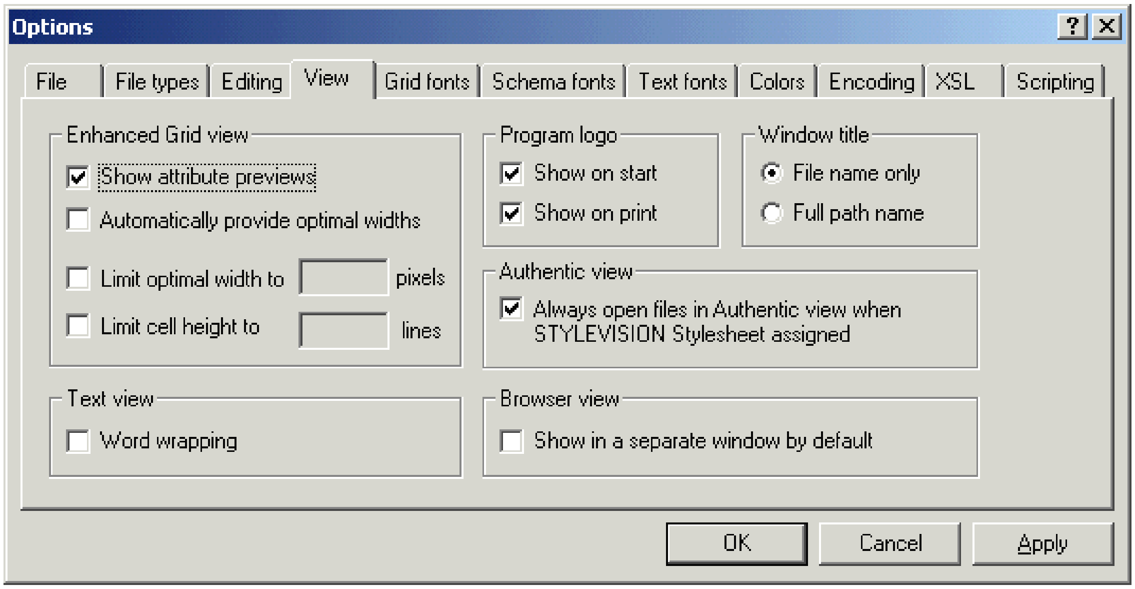
The structured text mode attempts to format the clipboard contents as a table, for use in a spreadsheet or database application. This option does not affect the internal clipboard format that XMLSpy uses for copying and pasting.

Table view

You can also control, how XMLSpy decides when to display repeating elements in the Table View.

View

The **View** tab enables you to customize the XML documents presentation in XMLSpy.



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

Text view

□ Word wrapping

Browser view

□ Show in a separate window by default

Enhanced Grid View

Collapsed elements in the Enhanced Grid View can be displayed in preview form. This displays the attributes and their values in gray in the same line as the element. You can automatically apply the Optimal Widths command while editing, and limit the maximum cell width and height.

Text View

Turn the Text View word-wrapping support on or off.

Program logo

You can turn off the splash screen upon program startup to speed up the application.

Window title

The window title for each document window can contain either the file name only or the full path name.

Authentic View

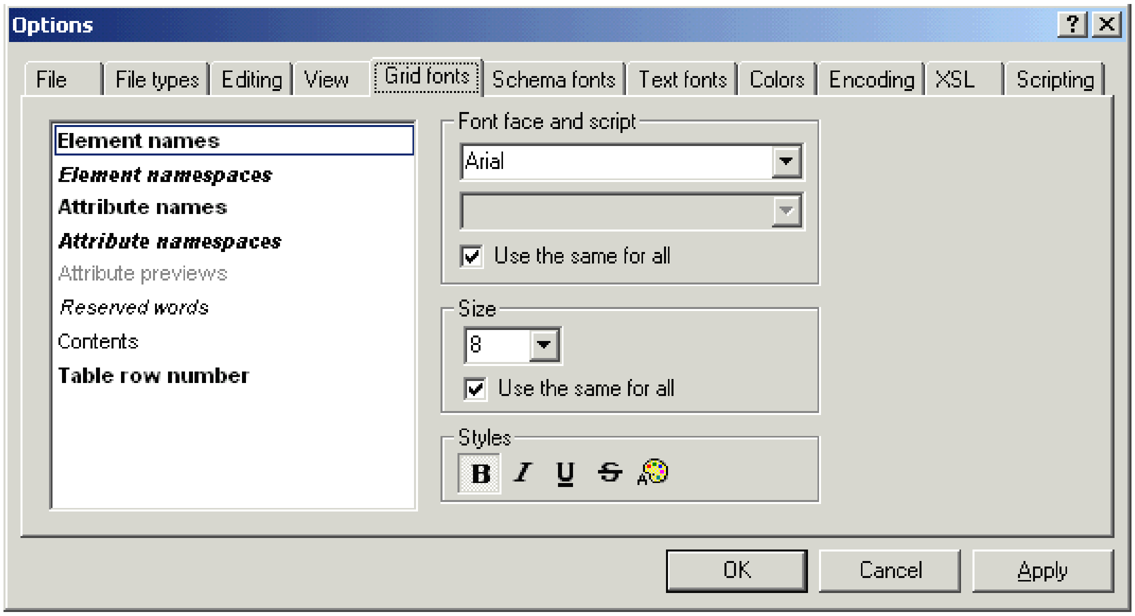
XML files based on a **StyleVision Power Stylesheet** are automatically opened in the Authentic View when this option is active.

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.

Grid fonts

The **Grid fonts** tab allows you to customize the appearance of text in Grid View.



Font face and script

You can select the font face and size to be used for displaying the various items in Enhanced Grid View. The same fonts are also used for printing, so only TrueType fonts should be selected.

Size

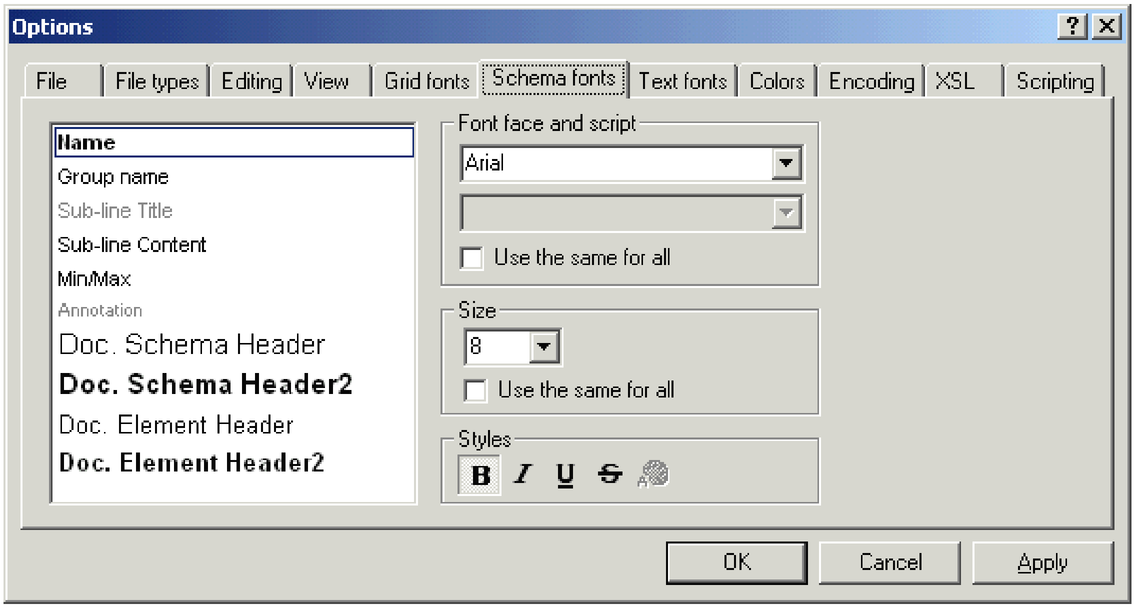
Select the required size. If you want to use the same font size for all items, check the **Use the same for all** check box.

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.

Schema fonts

The **Schema fonts** tab enables you to customize the appearance of text in the Schema/WSDL View.



Options

Name
Group name
Sub-line Title
Sub-line Content
Min/Max
Annotation
Doc. Schema Header
Doc. Schema Header2
Doc. Element Header
Doc. Element Header2

Font face and script: Arial

□ Use the same for all

Size: 8

□ Use the same for all

Styles: **B** *I* U S

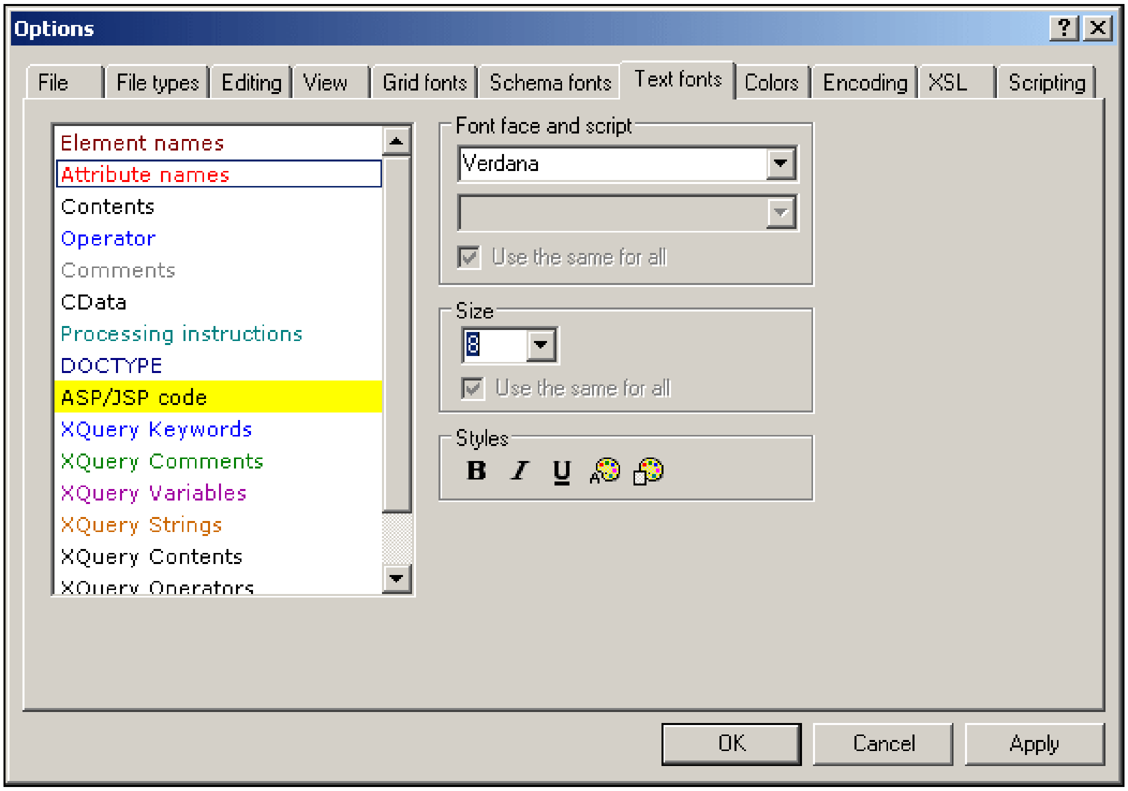
Font face and script
You can select the font face and size to be used for displaying the various items in the Schema/WSDL Design view. The same fonts are used when printing and creating schema documentation, so only TrueType fonts should be selected. Components prefixed with "Doc." are used in the schema 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 Al" 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.

Text fonts

The **Text fonts** tab you to customize the appearance of text in the Text View.



Options

Font face and script: Verdana

☑ Use the same for all

Size: 8

☑ Use the same for all

The types listed in the left hand pane are XML node types, ASP/JSP code, and XQuery grammar components. You can choose the common font face, style and size of all text that appears in Text View. Note that the same font, style, and size is used for all text types. Only the text color and background color can be changed for individual text types. This enables the syntax coloring feature.

Colors

The **Colors** tab enables you to customize the background colors used in the Table View of Grid View.

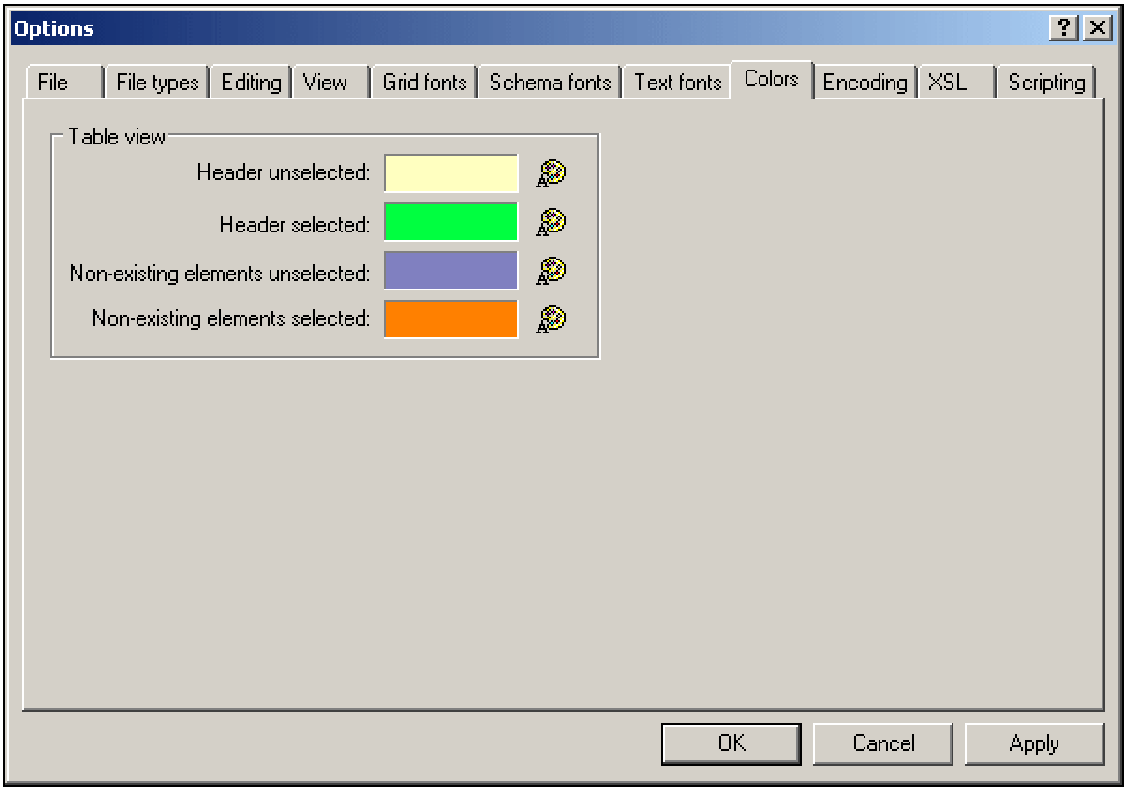


Table view

Header unselected: 🟨

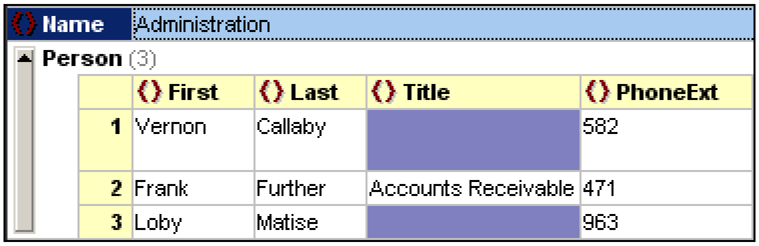
Header selected: 🟩

Non-existing elements unselected: 🟪

Non-existing elements selected: 🟧

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.

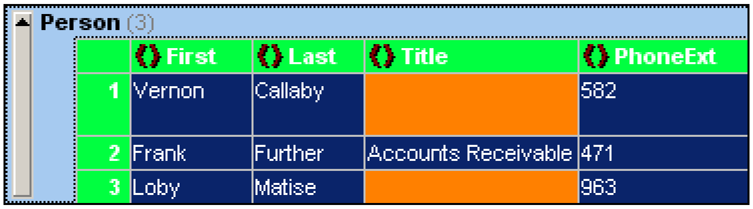


Name Administration

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.

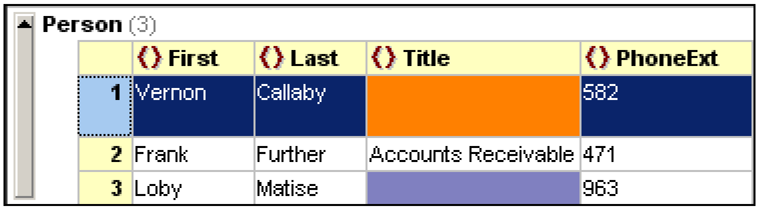


Person (3)

	First	Last	Title	PhoneExt
1	Vernon	Callaby	🟧	582
2	Frank	Further	Accounts Receivable	471
3	Loby	Matise	🟧	963

Non-existent Elements

When a cell in the table is empty, the element or attribute represented by that cell does not exist in the XML document. Such non-existent cells can be given different background colors when selected and unselected. This is shown in the screenshot below, in which the first row is selected.



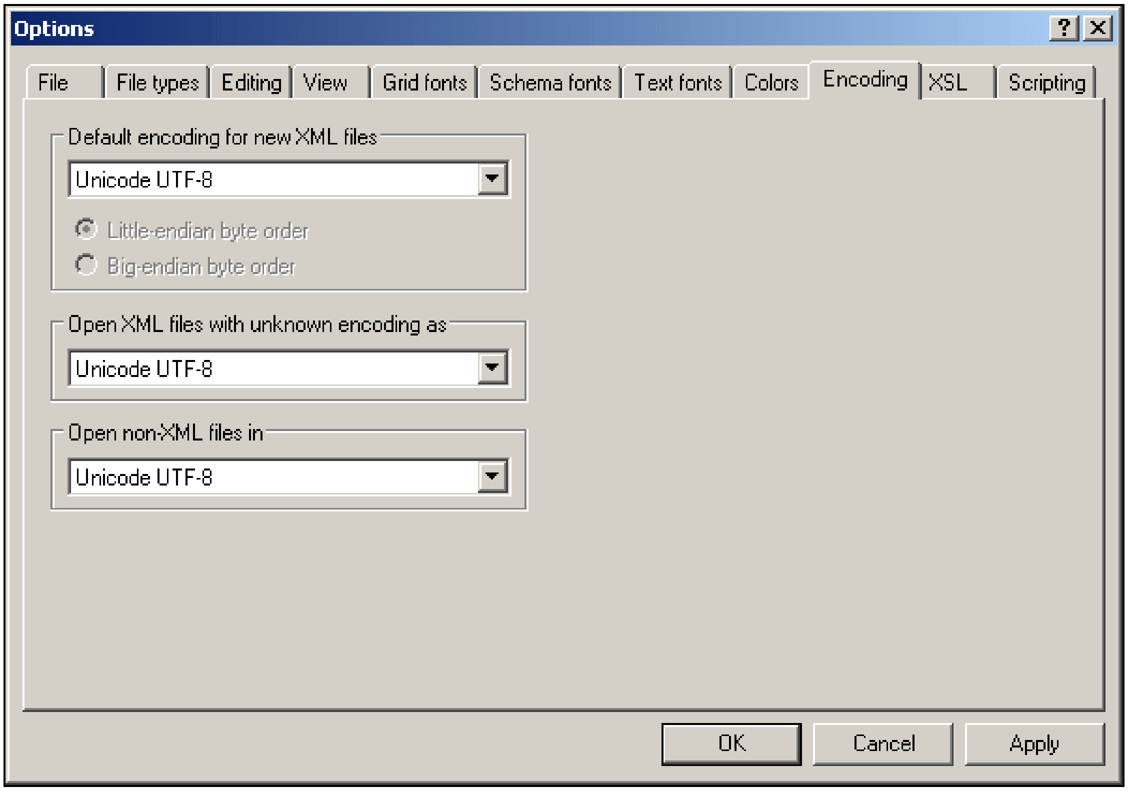
Person (3)

	() 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.

Encoding

The **Encoding** tab specifies options for file encodings.



Default encoding for new XML files: Unicode UTF-8

● Little-endian byte order

○ Big-endian byte order

Open XML files with unknown encoding as: Unicode UTF-8

Open non-XML files in: Unicode UTF-8

Default encoding for new XML files

The default encoding for new files can be pre-determined in the Settings dialog box so that each new document is automatically created with a proper XML-declaration and includes the encoding-specification that 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 for the XML files. The Default Encoding for New XML Files setting only works for files which do not have a StyleVision Power Stylesheet associated with it.

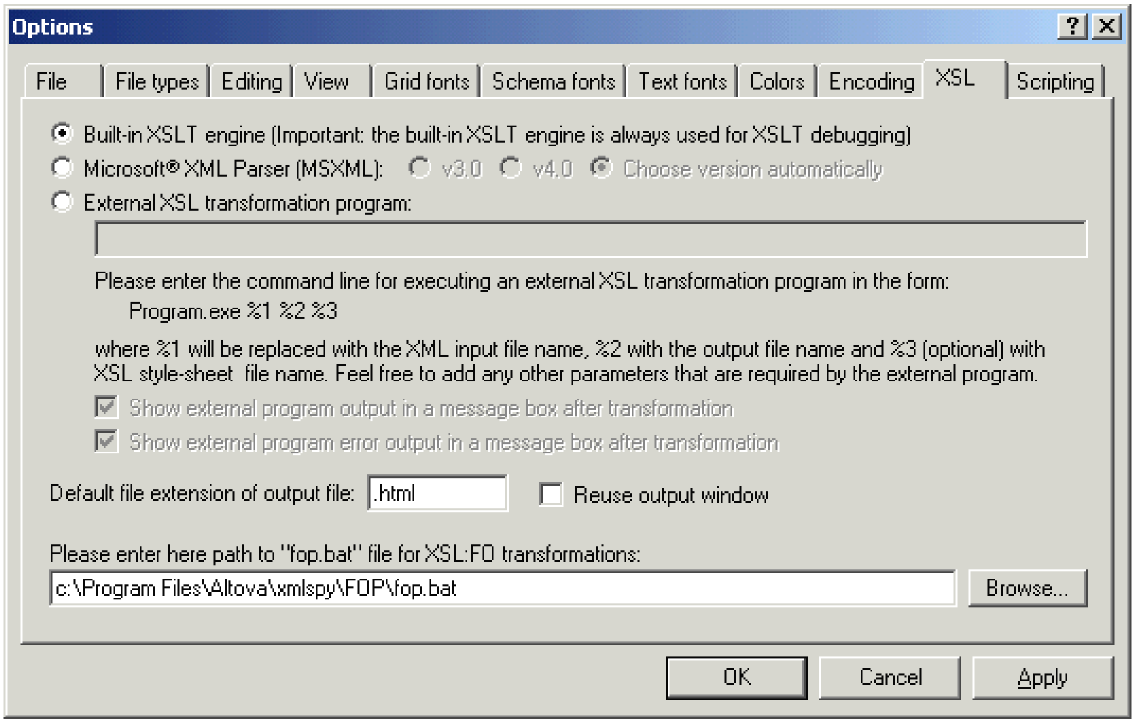
The encoding for existing files will, of course, always be retained and can only be changed with the **File | Encoding** command.

Open XML files with unknown encoding as
You can select the encoding with which to open an XML file with no encoding specification or where the encoding cannot be detected.
Please note: XML files which have no encoding specification are correctly saved with a UTF-8 encoding.

Open non-XML files in
You can select the encoding of non-XML files that you wish to edit.

XSL

The **XSL** tab allows you to define options for XSLT transformations as well as the path to an FO processor.



● Built-in XSLT engine (Important: the built-in XSLT engine is always used for XSLT debugging)
○ Microsoft® XML Parser (MSXML): ○ v3.0 ○ v4.0 ● 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 a message box after transformation
☑ Show external program error output in a message box after transformation

Default file extension of output file: .html □ Reuse output window

Please enter here path to "fop.bat" file for XSL:FO transformations:
c:\Program Files\Altova\xmlspy\FOP\fop.bat

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 is always used (according to the value of the `version` attribute set on the `xsl:stylesheet` or `xsl:transform` element) for transformations and when using the XSLT/XQuery Debugger.

For transforming XML documents, you could use either the MSXML 3.0 or 4.0 parser (which is pre-installed) or any external XSLT processor of your choice. Note that parameters set in the XSLT Input Parameters dialog are passed to the internal Altova XSLT Engines only; they are not passed to any other XSLT Engine that you set up as the standard XSLT processor to be used in XMLSpy.

Use Microsoft XML Parser (MSXML) 3.0 or 4.0
If you are using Internet Explorer 5 or above (i.e. the MSXML module) for XSLT processing, you must specify if the resulting output is to be transformed as an XML-compliant document (such as XHTML or WML), or non-XML compliant text files. The "Choose version automatically" option is active by default. XMLSpy first tries to use version 4.0 and if not available version 3.0. You can however, manually choose either version by clicking the respective radio button.

External XSL transformation program

If you are using an external XSL processor, you must specify a command-line string that is to be executed by XMLSpy in order to initiate the XSL Transformation. In this case, use the following variables to build your command-line string:

`%1` the XML instance document that is to be processed
`%2` the output file that is to be generated
`%3` the XSL Stylesheet to be used (optional, if the document contains an `<?xsl-stylesheet ...?>` reference)

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
```

In order to set up XMLSpy to run the Saxon processor, select the External XSL Transformation Program radio button, and enter the following line in the text box:

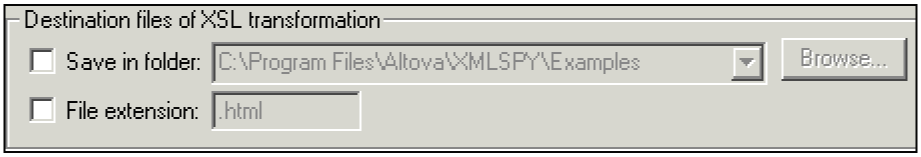
```
c:\saxon\saxon.exe -o %2 %1 %3 parameter-name=parameter-value
```

You can also choose to show the result of the external program's output and error messages in a message box within XMLSpy.

Reuse output window

When disabled, the reuse output window option only takes effect when you **also** disable the "Save in folder" output file path in the folder properties of the respective XML file in a XMLSpy project.

Folder properties window:



Destination files of XSL transformation

□ Save in folder: C:\Program Files\Altova\XMLSPY\Examples

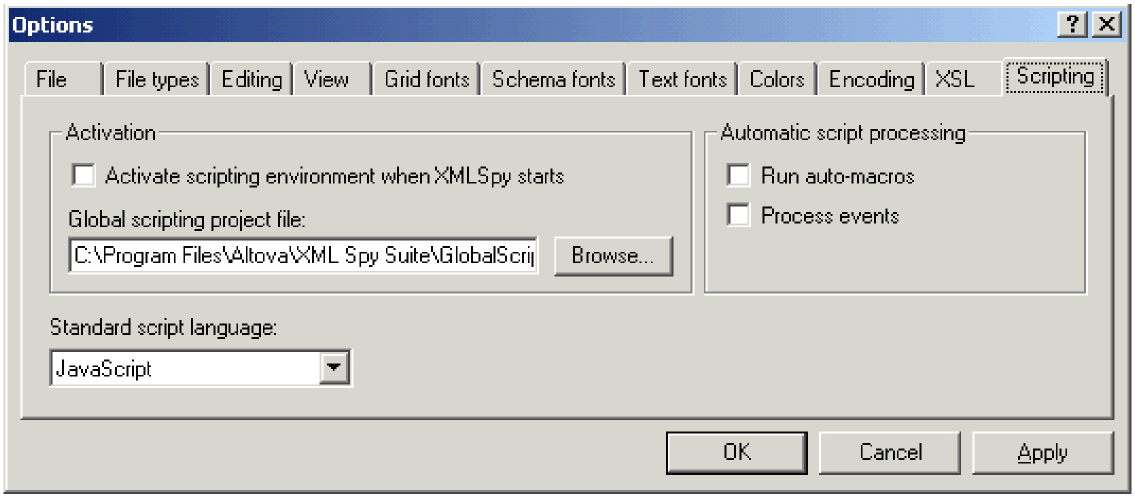
□ File extension: .html

FO processor

If you wish to use the FO transformation command in the interface, you must specify the path to the executable of the FO processor in this dialog.

Scripting

The **Scripting** tab allows you to select the scripting environment and other scripting specific parameters.



Activation

□ Activate scripting environment when XMLSpy starts

Global scripting project file: C:\Program Files\Altova\XML Spy Suite\GlobalScri

Automatic script processing

□ Run auto-macros

□ Process events

Standard script language: JavaScript

Activation

Define if the scripting environment is to be active when XMLSpy starts. Click the **Browse...** button to select the scripting project file.

Standard script language

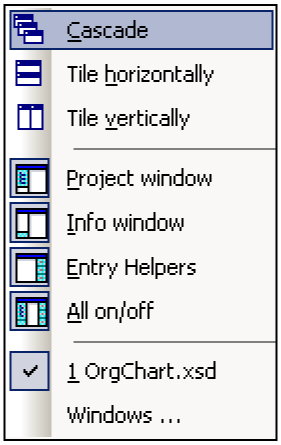
Select the scripting language you want to use (JavaScript or VBScript).

Automatic script processing

This section defines if auto-macros have the right to run, and if event processing is to be enabled.

11.13 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.

11.13.1 Cascade

This command rearranges all open document windows so that they are all cascaded (i.e. staggered) on top of each other.

11.13.2 Tile horizontally

This command rearranges all open document windows as **horizontal tiles**, making them all visible at the same time.

11.13.3 Tile vertically

This command rearranges all open document windows as **vertical tiles**, making them all visible at the same time.

11.13.4 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.

11.13.5 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.

11.13.6 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.

11.13.7 All on/off

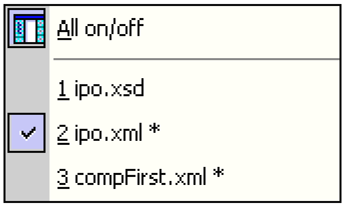
This command lets you switch all dockable windows on, or off:

- the Project Window
- the Info Window
- the three Entry-Helper Windows

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.

11.13.8 Currently open window list

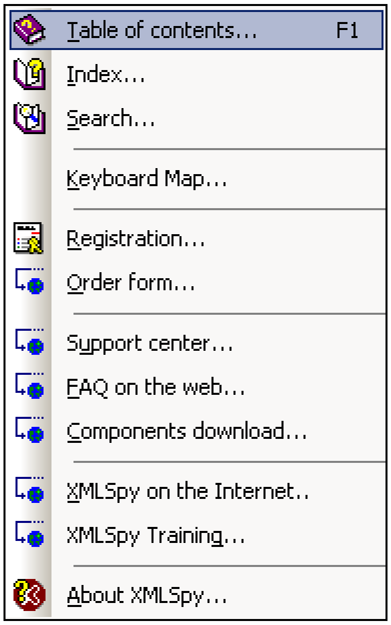
This list shows all currently open windows, and lets you quickly switch between them.



You can also use the Ctrl-TAB or CTRL F6 keyboard shortcuts to cycle through the open windows.

11.14 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.

11.14.1 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.

11.14.2 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.

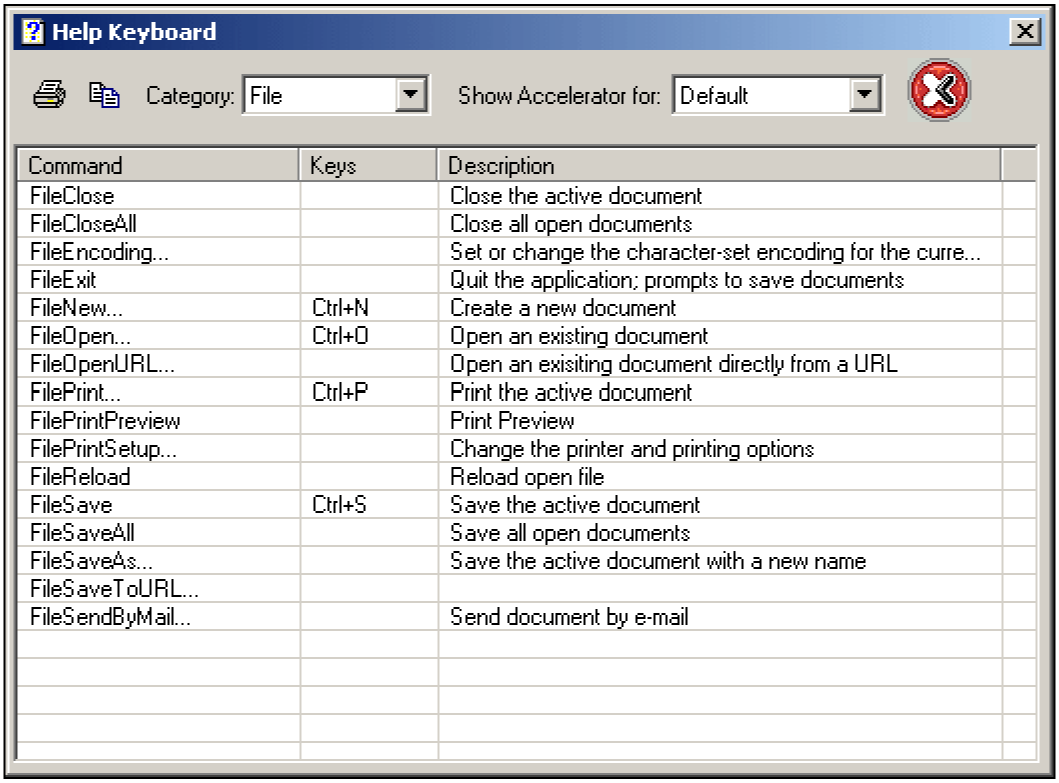
11.14.3 Search...

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.

11.14.4 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.



Category: File

Show Accelerator for: Default

Command	Keys	Description
FileClose		Close the active document
FileCloseAll		Close all open documents
FileEncoding...		Set or change the character-set encoding for the curre...
FileExit		Quit the application; prompts to save documents
FileNew...	Ctrl+N	Create a new document
FileOpen...	Ctrl+O	Open an existing document
FileOpenURL...		Open an exisiting document directly from a URL
FilePrint...	Ctrl+P	Print the active document
FilePrintPreview		Print Preview
FilePrintSetup...		Change the printer and printing options
FileReload		Reload open file
FileSave	Ctrl+S	Save the active document
FileSaveAll		Save all open documents
FileSaveAs...		Save the active document with a new name
FileSaveToURL...		
FileSendByMail...		Send document by e-mail

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.

11.14.5 Registration...

When you start XMLSpy for the first time, you are automatically presented with the Registration dialog box, which lets you register your software product in order to be eligible for technical support and activate your license, which is done by entering a unique key-code to unlock the software.

FREE Evaluation Version

If you have downloaded the XMLSpy from our web server and would like to activate your **FREE** 30-day evaluation version, please enter your name, company, and e-mail address and click on the "Request FREE evaluation key..." button. XMLSpy then uses your Internet connection to transmit the information you have just entered to our web server, where a personal unique evaluation license will be generated for you. The license key-code, which is necessary to unlock your software, will then be sent to the e-mail address you have entered - it is therefore important that you enter your **real e-mail address** in the registration dialog box!

Once you have clicked the request button, please go to your favorite mail software and retrieve the license key-code from our e-mail message, which you should be receiving in a matter of a few minutes (depending on transient Internet conditions).

If you requested a key-code and it didn't arrive in a short space of time, the process may have failed due to Firewall restrictions in your network. If this is the case, please send a short message with your information via e-mail to our website and our support staff will generate a key-code for you manually.

When you have received your evaluation key-code, please enter it into the key-code field in the registration dialog box and click **OK** to start working with XMLSpy.

Whenever you want to place an order for a licensed version of XMLSpy, you can also use the **Order license key...** button in the registration dialog box or the Order form menu command to proceed to the Secure Online Shop on the Internet.

Licensed Version

If you have purchased a *single-user* license for XMLSpy, you will receive an e-mail message from us that contains your license-data and includes your name, company and key-code. Please make sure that you enter **all fields** from your license e-mail into the registration dialog box. The key-code will only be able to unlock your software installation, if the entries in the name and company fields match the name and company entered into our order form.

If your company has purchased a *multi-user* license for XMLSpy, you will receive an e-mail message from us that contains your license-data and includes your company name and key-code.

Please make sure that you enter the company name and key-code from your license e-mail into the registration dialog box and also enter your personal name into the name field. The key-code will only be able to unlock your software installation, if the value in the company field matches the company name entered into our order form.

Please note:
The XMLSpy License-Agreement does not allow you to install more than the licensed number of copies of XMLSpy on the computers in your organization (per-seat license).

11.14.6 Order form...

When you want to place an order for a licensed version of XMLSpy, use this command or the "Order license key..." button in the registration dialog to proceed to the Secure Online Shop on the Internet, where you can choose between different single- and multi-user license packs.

Once you have placed your order, you can choose to pay by credit card, send a check by mail, or use a bank wire transfer.

11.14.7 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.

11.14.8 FAQ on the web

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.

11.14.9 Components download

The Components download option currently lets you to download the latest Microsoft XML Parser, as well as an alternate XSLT Transformation System, and will be expanded in the future.

11.14.10On 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.

11.14.11Training...

This command takes you directly to the Altova web-server http://www.altova.com where you can find out about our authorized Training Partners who provide courses on using XMLSpy and Advanced XML Application Development (AXAD).

11.14.12About...

This command shows the XMLSpy splash screen and copyright information dialog box, which includes the XMLSpy logo.

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.

Altova XMLSpy Professional Edition User Manual

Programmers' Reference

Programmers' Reference

XMLSpy as an Automation Server

XMLSpy is an Automation Server. That is, it is an application that exposes programmable objects to other applications (called Automation Clients). As a result, an Automation Client can directly access the objects and functionality that the Automation Server makes available. This is beneficial to an Automation Client because it can make use of the functionality of XMLSpy. For example, an Automation Client can use the XML validation functionality of XMLSpy. Developers can therefore improve their applications by using the ready-made functionality of XMLSpy.

The programmable objects of XMLSpy are made available to Automation Clients via the **XMLSpy API**, which is a COM API. The object model of the API and a complete description of all available objects are provided in this documentation (see the section XMLSpyAPI). The API can also be used with a Java implementation. A description of how this can be done is given in the chapter Using the XMLSpy API with Java.

Customizing XMLSpy, and modifying 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 XMLSpy API. To aid you in carrying out these tasks efficiently, XMLSpy offers you an in-built Scripting Environment. A complete description of the functionality available in the Scripting Environment and how the environment is to be used is given in the Scripting section of this documentation.

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. These external scripts would again make use of the XMLSpy API to carry out these tasks. For a description of the XMLSpy API, see the section XMLSpyAPI.

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 Plug-ins.

About Programmers' Reference

The documentation contained in the Programmers' Reference for XMLSpy consists of the following sections:

- Release Notes, which lists changes since the previous version
- Scripting: a user reference for the Scripting Environment available in XMLSpy
- XMLSpy Plug-ins: a description of how to create plug-ins for XMLSpy
- XMLSpy API: a reference for the XMLSpy API
- Using XMLSpy API with Java: a reference for using the XMLSpy API with Java

1 Release Notes

For each release of the XMLSpy API, important changes since the previous release are listed below.

Automation Interface for XMLSpy 2006sp2 - type library version 1.5
Changes since XMLSpy API ver 2006:

- The `CodeGeneratorDlg` interface now has the properties: `CompatibilityMode`, `CPPSettings_GenerateVC6ProjectFile`, `CPPSettings_GenerateVSProjectFile`. These properties generate project files for Visual Studio 2005.

Automation Interface for XMLSpy 2006 - type library version 1.5
Important changes since the previous release (XMLSpy API 2004R4) are as follows:

- The `Application` interface has the following methods: `ReloadSettings`, `RunMacro`, `ScriptingEnvironment`.
- The `Document` interface has the following additional methods: `GenerateSampleXML`, `SetExternalIsValid`, `UpdateXMLData`.
- The `Document` interface has the additional event: `OnBeforeValidate`.
- The `XMLData` interface has the following additional declarations: `CountChildren`, `CountChildrenKind`, `HasChildrenKind`, `GetChild`, `GetChildKind`.
- A new interface `GenerateSampleXMLDlg` has been added.

Automation Interface for XMLSpy 2004R4 - type library version 1.4
Important changes since the previous release (XMLSpy API 2004R3) are as follows:

Authentic View

- `AuthenticView`, `AuthenticRange` and `AuthenticDataTransfer` interfaces replace `DocEdit`, `DocEditSelection` and `DocEditDataTransfer`.
- The `AuthenticView` and `AuthenticRange` interfaces now supersede the old `DocEdit` interface.
- The `DocEdit` interfaces have been renamed `OldAuthentic` and are now obsolete.
- The `DocEditSelection` interface has been renamed `AuthenticSelection` and is now obsolete.
- The `DocEditDataTransfer` interface has been renamed `AuthenticDataTransfer`.
- The functionality of now obsolete interfaces are still available. However, usage of the new interfaces is strongly recommended.

Improved events
All events are available as connection point events to allow for easy and flexible integration of external clients using VBA, C++, VBScript, JScript or Perl. New events supporting the document life cycle have been added. All events pass event context information as parameters. Many events support cancellation of the signaled operation. See the section Overview of Events for more details.

Extension of Document interface

- Supports generation of program code (C++, C# and Java) for schema definitions
- Supports generation of MSWord and HTML documentation for schema definitions
- Supports XSL-FO transformation
- Extension of standard properties for documents

- Direct access to data root element of XML documents

Data import and export

- Choice of database for database import.
- Import of hierarchical data, which was previously done using a hand-written SHAPE statement with ImportFromDatabase, now requires a schema file and the use of ImportFromSchema.

Automation Interface for XMLSpy 2004R3 - type library version 1.3

Important changes since XMLSpy API ver 5 rel 4:

Authentic View

- Completely new interfaces `AuthenticView` and `AuthenticRange` to work with Authentic View.

2 Scripting

XMLSpy provides a **Scripting Environment** within the XMLSpy IDE. The Scripting Environment can be used to:

- Graphically design Forms for user input and for output
- Create, manage, and store scripts for these Forms
- Create scripts for application Event Handlers and for Macros.

Scripts access and interact with objects in the XMLSpy API, which is a COM API. They thus allow you to modify and add functionality to your installation of XMLSpy. Programming languages that can be used in the Scripting Environment are **JavaScript** and **VBScript**.

Forms
In the Scripting Environment, you 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 the text input field so as to take an input variable; and an event handler can be associated with a button to start execution of the functionality, which is available in the XMLSpy API. For details, see Creating a Form.

Global Declarations
In the Scripting Environment, a set of Global Declarations contains the variables and functions that will be used by Forms, Event Handlers, and Macros. The functions make use of the XMLSpy API to access XMLSpy functionality.

Event handling
Using the Scripting Environment, you can easily customize XMLSpy by writing scripts for a variety of available events. You can control events that occur both within Forms (Form events) and within the general XMLSpy interface (application events).

Scripting code for an event is executed immediately upon the triggering of an event. Most events have parameters which provide detailed information on the event. The return value from the script typically instructs the application about how to continue its own processing (for example, the application may not allow editing). For details, see Creating an Event Handler.

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. Please see Writing a Macro for a simple example of creating a Macro. Also see Calling macros for a description of the ways in which a Macro can be called.

2.1 Overview

This overview of the Scripting Environment discusses the following:

- The relationship between scripting projects and XMLSpy projects
- How and when Forms, Event Handlers and Macros are executed within XMLSpy
- How settings for the Scripting Environment are to be made (in XMLSpy)

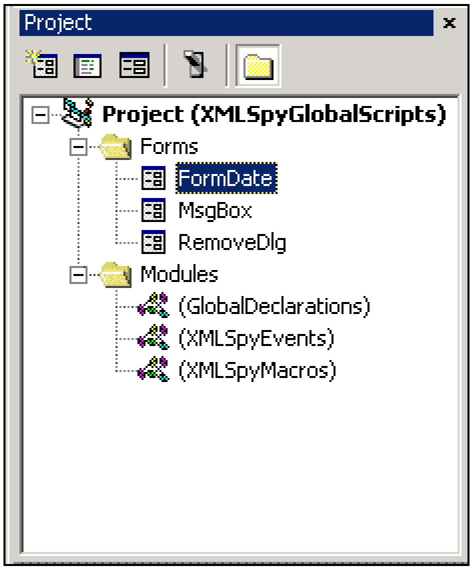
2.1.1 Scripting projects

Working with scripting projects

- In the Scripting Environment, all scripts and scripting information are stored in **scripting projects** (**`.prj` files**).
- After you have created and saved a scripting project, you must assign it to an XMLSpy project. The scripts in the scripting project can then be used when the XMLSpy project to which it has been assigned is opened in XMLSpy.
- Only one scripting project can be assigned to an XMLSpy project at a time.
- You can assign a scripting project to more than one XMLSpy project.
- An XMLSpy project uses the scripts in the global scripting project and the scripts in the scripting project assigned to it.
- If you wish to make scripts applicable to the entire XMLSpy application, you must modify the global scripting project.
- The scripts in the global scripting project apply to the entire XMLSpy application even when no XMLSpy project is open.

The contents of a scripting project and how scripting projects are assigned to and unassigned from XMLSpy projects are described below.

Content of a scripting project
The Scripting Environment screenshot below shows the content of a typical scripting project.



The following points should be noted:

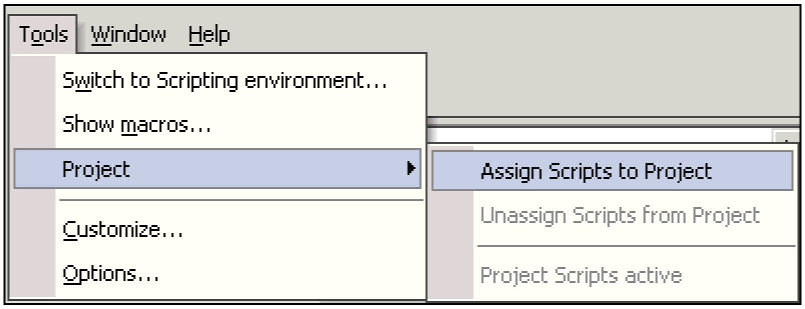
- A scripting project consists of two components: a **Forms** component and a **Modules** component. The Forms component contains all the Forms defined for that scripting project. The Modules component comprises three modules: **Global Declarations**,

XMLSpy Events, and **XMLSpy Macros**.The Global Declarations module contains definitions for variables and functions used in that scripting project. The XMLSpy Events module contains event handler scripts for that scripting project. The XMLSpy Macros module contains the macros that have been defined for that project.

- A scripting project, called `GlobalScripts.prj`, is installed with XMLSpy. This scripting project contains default global declarations and template scripts for standard XMLSpy event handlers and macro tasks.
- Forms, events, and macros defined in the global scripting project can be accessed from within any XMLSpy project or XML file.
- In addition to the global scripting project, any number of additional scripting projects can be created within the Scripting Environment and saved.

Assigning scripting projects
A scripting project is assigned to an **XMLSpy project** from within the XMLSpy interface with the required XMLSpy project open.

To assign and unassign a scripting project to a project, use the **Tools | Project** submenu (see screenshot below). (These menu items are not enabled if there is no XMLSpy project active currently.)


Tools
Window
Help
Switch to Scripting environment...
Show macros...
Project
Customize...
Options...
Assign Scripts to Project
Unassign Scripts from Project
Project Scripts active

To assign a scripting project, click **Assign Scripts to Project** and select the scripting project (`.prj` file) from the standard file-selection dialog that appears. The scripting project is loaded, and you are able to edit scripts in it if you switch to the Scripting Environment.

To unassign the current scripting project, select **Unassign Scripts from Project**. The scripting project is unassigned, and the Scripting Environment closes the scripting project.

The following points should be noted:

- For a given XMLSpy project, scripts from the scripting project specific to that XMLSpy project as well as scripts from the global scripting project can be accessed.
- Only **one** scripting project can be assigned to an XMLSpy project at a time.

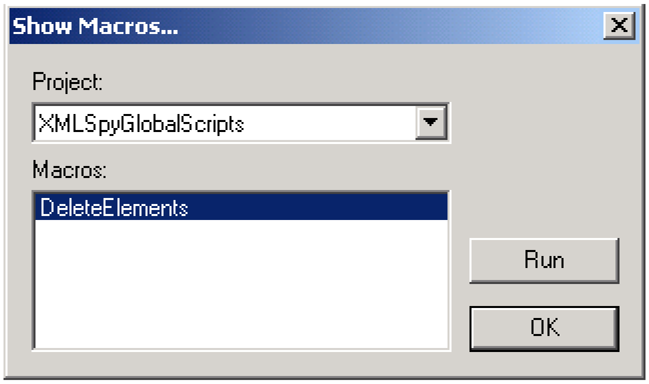
Making a scripting project active
A scripting project is said to be active when it is the active project **in the Scripting Environment**. (More than one scripting project can be open in the Scripting Environment, but only one is active.)

You can activate and deactivate the project-specific scripting project by toggling the **Project Scripts active** command on and off. Note that making the scripting project inactive **does not unassign** it; it merely deactivates the sripting project in the Scripting Environment.

2.1.2 Running Forms, Event Handlers, and Macros

Forms, Event Handlers, and Macros that have been defined in the Scripting Environment are all executed within the XMLSpy interface. 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 module 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 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 | Show macros...**. This pops up the Show Macros dialog:



In the Project combo box, select either the global scripting project or the scripting project currently assigned to the XMLSpy project (only these two scripting projects are available as options). The macros defined in the scripting project you select appear in the Macros window. Select the macro you wish to run, and click **Run**.

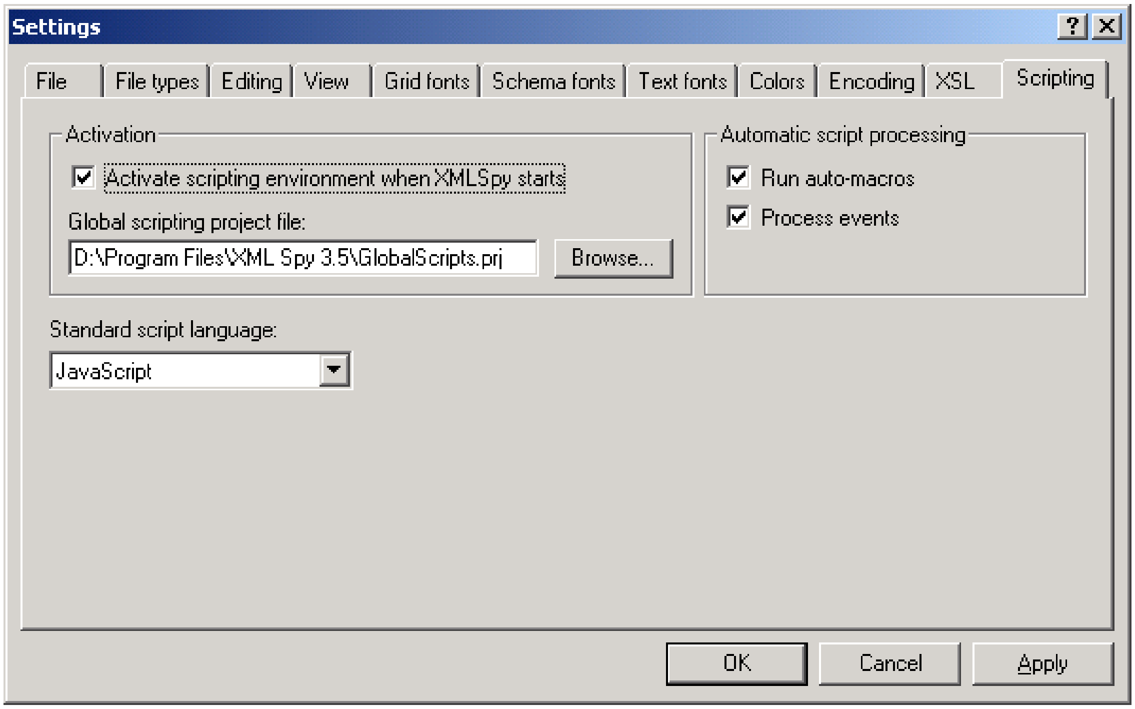
You can also customize your Tools menu to display a list of Macros (**Tools | Show Macros...**). A Macro in this list can subsequently be run by selecting it. See Calling macros for details.

Also see:

- Scripting Settings for information about configuring the XMLSpy interface for scripting.
- Calling macros for details about the various ways Macros can be called.
- The examples of how to create Forms, Global Declarations, Event Handlers, and Macros.

2.1.3 Scripting Settings

The screenshot below shows the Settings dialog for the Scripting Environment of XMLSpy:



Activation

☑ Activate scripting environment when XMLSpy starts

Global scripting project file: D:\Program Files\XML Spy 3.5\GlobalScripts.prj

Automatic script processing

☑ Run auto-macros

☑ Process events

Standard script language: JavaScript

Activate scripting environment when XMLSpy starts
If this checkbox is selected, the Scripting Environment is opened as a separate window when XMLSpy starts. Otherwise, the Scripting Environment is not activated until the user selects **Tools | Switch to scripting environment** for the first time.

Global scripting project file
This entry specifies the global scripting project file. A sample global scripting project file is installed with your XMLSpy application. If this file does not exist when the Scripting Environment starts, a new scripting project with the specified name will be created at the specified location.

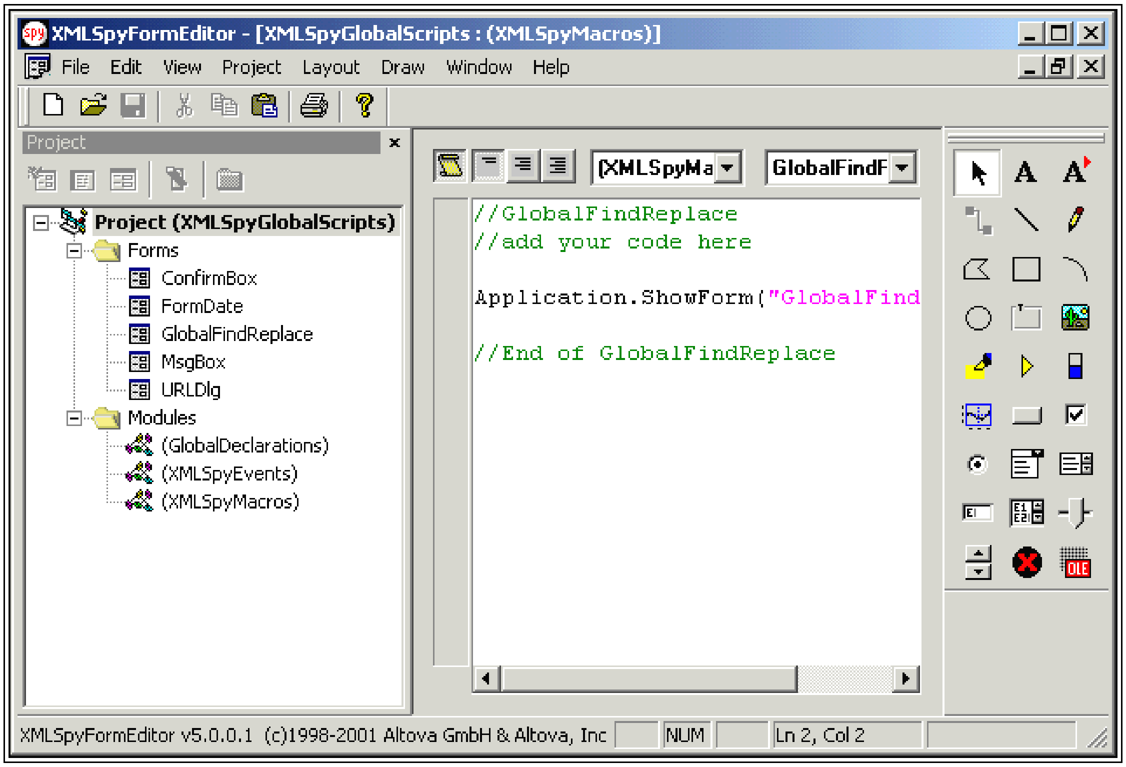
Run auto-macros
Enables or disables the execution of auto-macros. Currently Autorun from the global scripts is the only supported automatic macro. Autorun executes when the scripting environment starts.

Process events
Enables or disables the execution of event handlers. If this check box is not ticked, then Event Handlers in the scripting projects will not be executed.

Standard script language
Every new scripting project created in XMLSpy will propose this scripting language as the default scripting language..

2.2 The Scripting Environment GUI

The Scripting Environment GUI (shown below) is launched from XMLSpy by selecting **Tools | Switch to scripting environment...**. Alternatively, you can specify in XMLSpy (**Tools | Options | Scripting**) that the Scripting Environment be activated automatically each time XMLSpy starts.



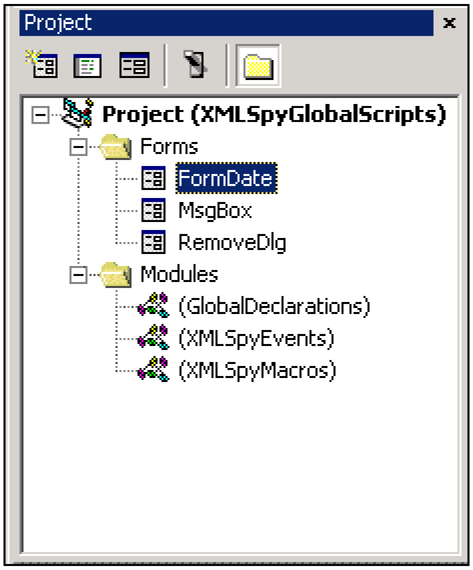
```
//GlobalFindReplace
//add your code here

Application.ShowForm("GlobalFind

//End of GlobalFindReplace
```

The Scripting Environment window has three parts: a Project Window, Main Window, and Form Object Bar. Each of these parts is described individually below. When a Form is active in the Main Window, the Scripting Environment becomes an effective graphical editor, called the **Form Editor**. The special features of the Form Editor are described separately below.

2.2.1 Project Window

Project Window
The Project Window displays all open scripting projects in a tree view. Only one scripting project can be active at a time. The title of the active project is displayed in bold.

To set a scripting project as the active project, right-click the scripting project and select **Set as active Project**. Closing a project removes it from the Project Window. If a project is not the active project, closing it automatically makes it the active project before it is closed—therefore, after it is closed no project is active.



Double-clicking a Form or Module in the Project Window, opens that Form or Module in the Main Window, where you can graphically design the Form or edit the script.

Icons

The following icons are in the Project Window's title bar:

New Form: Adds a new form to the Forms folder.

View Code: Displays the selected Form or the code of a Module. The code of the selected Form is displayed only if **Layout | Edit Script** has been toggled on for that Form. Otherwise the editable (if **Layout | Edit View** is selected) or non-editable (if neither **Layout | Edit Script** nor **Layout | Edit View** is selected) view of the Form is displayed. This icon is enabled when a Form or Module is selected.

View Form: Displays the selected Form. The editable view of the selected Form is displayed only if **Layout | Edit View** has been toggled on for that Form. Otherwise the code (if **Layout | Edit Script** is selected) or non-editable view (if neither **Layout | Edit View** nor **Layout | Edit Script** is selected) of the Form is displayed. This icon is enabled when a Form is selected.

Run Form: This icon is enabled when a Form is selecteds.

Toggle Folders: Toggles the Forms and Modules level of the tree on and off.

Context menus

Two context-sensitive popup menus are available within the Project Window. The first appears when you right-click the title of a project. The second appears when you right-click a Form or Module. These popup menus give you access to commands available in the **File** and **Project** menus.

2.2.2 Main Window

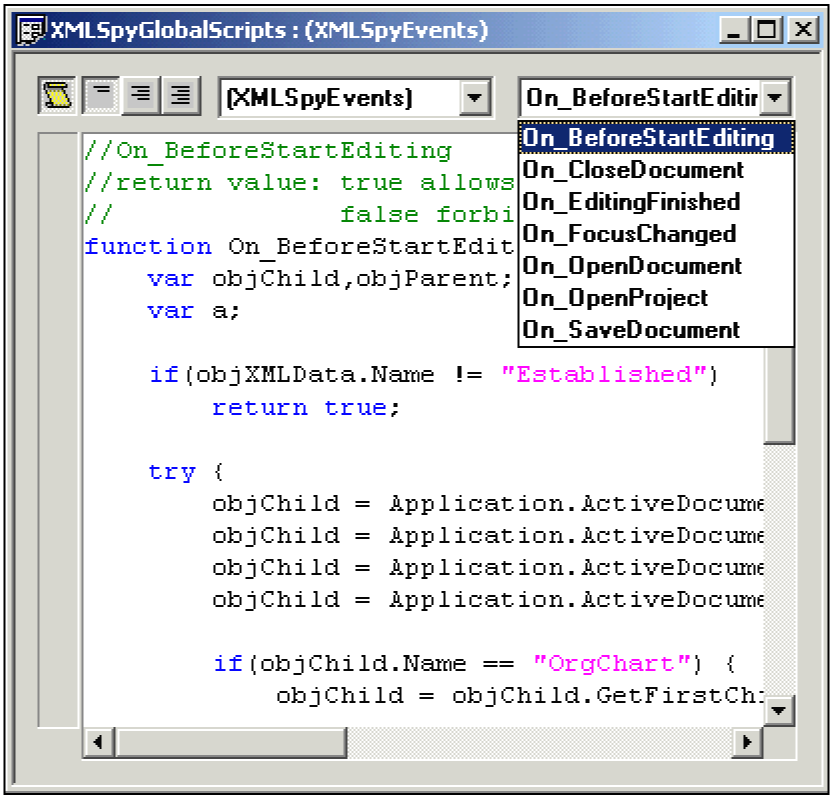
The Main Window is the area in the GUI where individual Forms and Modules are displayed in separate windows. It is in these windows that you design Forms and edit scripts.

Forms

- Each Form is displayed in a separate window.
- When the Scripting Environment GUI has a Form window active, it is called a Form Editor. For details of usage, see The Form Editor.
- A Form is displayed as the graphical dialog box presented to the user (Design View). If **Layout | Edit View** is toggled on for a given Form, then that Form is editable. Otherwise the Design View is non-editable.
- Additionally, if **Layout | Edit Script** has been toggled on for a Form, then the script underlying that Form is displayed in the Form window. Clicking the Close Script View icon switches back to the Design View (editable or non-editable according to whether **Layout | Edit View** is toggled on or off, respectively).
- Note that the **Layout | Edit View** and **Layout | Edit Script** switches are specified separately for each Form.

Modules

- Each Module (Global Declarations, (Application) Events, and Macros) opens in a separate window.
- Within a Module window, the combo box on the left indicates which module is selected (in the screenshot below, the combo box shows that the Events Module is the selected Module). To select a specific item (Global Declaration, Event-Handler, or Macro), click the required option in the drop-down menu of the combo box on the right.



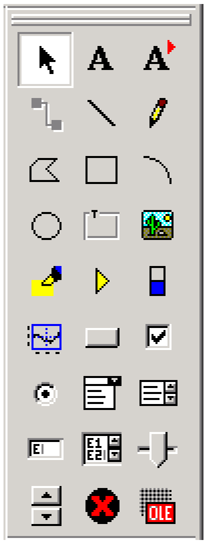
```
//On_BeforeStartEditing
//return value: true allows
//              false forbi
function On_BeforeStartEdit
    var objChild,objParent;
    var a;

    if(objXMLData.Name != "Established")
        return true;

    try {
        objChild = Application.ActiveDocume
        objChild = Application.ActiveDocume
        objChild = Application.ActiveDocume
        objChild = Application.ActiveDocume

        if(objChild.Name == "OrgChart") {
            objChild = objChild.GetFirstCh
```

Context menus

Right-clicking within the Script View of a Form or within a Module window pops up a context menu that gives you access to features that help you with your scripting.

2.2.3 Form Object Bar

The Form Object Bar is a tool palette that helps you add controls to your Forms. Select a Form Object from the bar, then place the pointer at the desired location in your Form and draw a rectangle to specify the size of the object.



The most commonly used objects are described below:

Static Text: Adds text fields such as captions or field descriptions.

Button: Adds a button. It is possible to assign bitmaps to these buttons.

Check Box: Adds a check box, which enables Yes–No type elements.

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.

Edit Box: Defines a single line of text.

Multi-edit Box: Defines multiple lines of text.

ActiveX: Allows the integration of an ActiveX control in the Form.

Setting Form Object properties and events

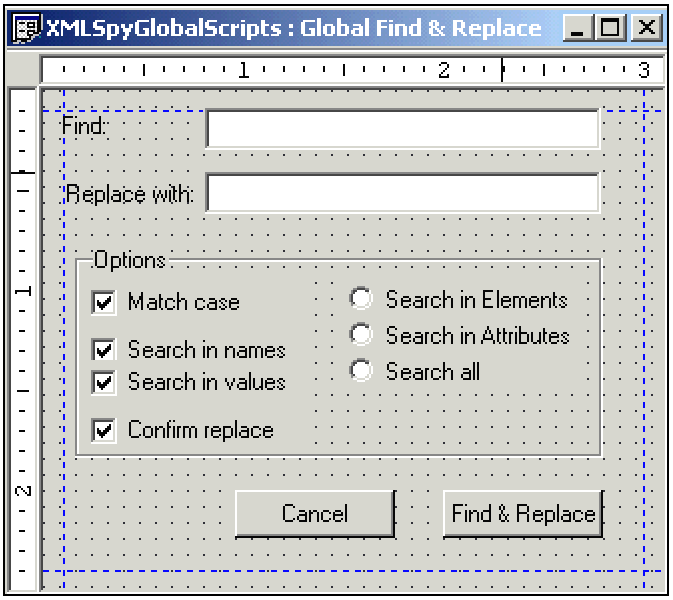
Each Form Object has its own set of properties and events. You can specify the settings of a Form Object in the Property Sheet for that object, as well as add some scripting code to the provided events. For details about how to do this, see The Form Editor.

2.2.4 The Form Editor

The Form Editor is the Scripting Environment used for editing a Form. A Form can be opened by double-clicking that Form's icon in the Project Window or by right-clicking a Form icon and selecting either **View Code** or **View Object**.

Form views

Each Form opens in a separate window (shown below) that can be resized and minimized. The two views in which a Form is displayed are Design View and Script View.



XMLSpyGlobalScripts : Global Find & Replace

Find:

Replace with:

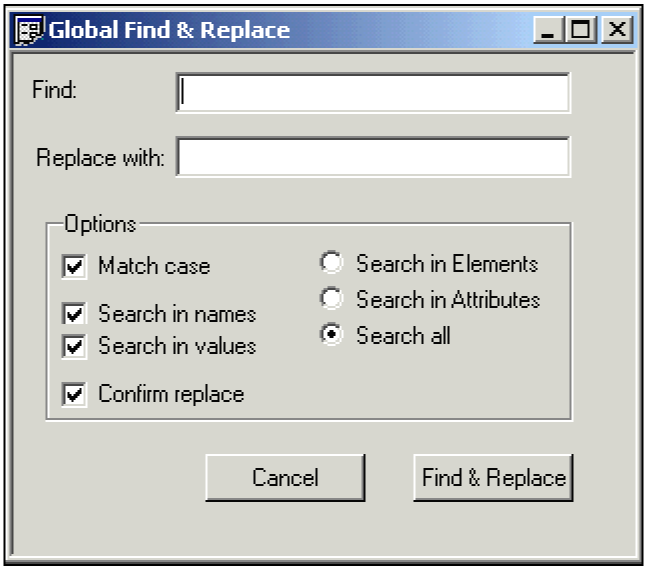
Options

☑ Match case
☑ Search in names
☑ Search in values
☑ Confirm replace

○ Search in Elements
○ Search in Attributes
○ Search all

Cancel | Find & Replace

The Design View shown above is editable. In order to make a Design View editable, you must toggle on the menu item **Layout | Edit View**. If this menu item is toggled off, then the Design View is uneditable and the Form appears as it will to the user (see screenshot below).



Global Find & Replace

Find:

Replace with:

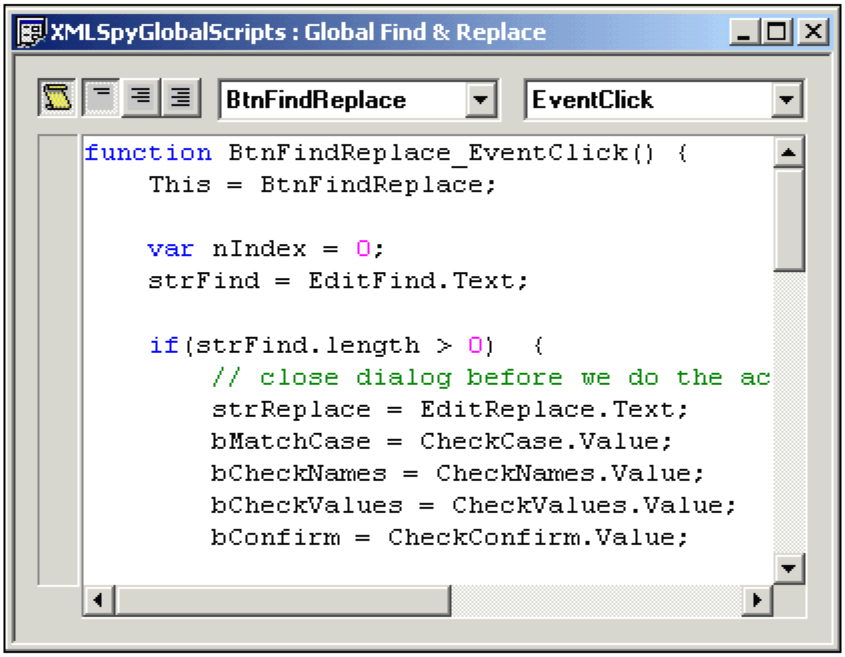
Options

☑ Match case
☑ Search in names
☑ Search in values
☑ Confirm replace

○ Search in Elements
○ Search in Attributes
● Search all

Cancel | Find & Replace

Additionally, a Form can be displayed in a Script View (shown below) by toggling on the menu item **Layout | Edit Script**. Closing the Script View switches the view back to Design View (editable or non-editable according to whether **Layout | Edit View** is toggled on or off, respectively).

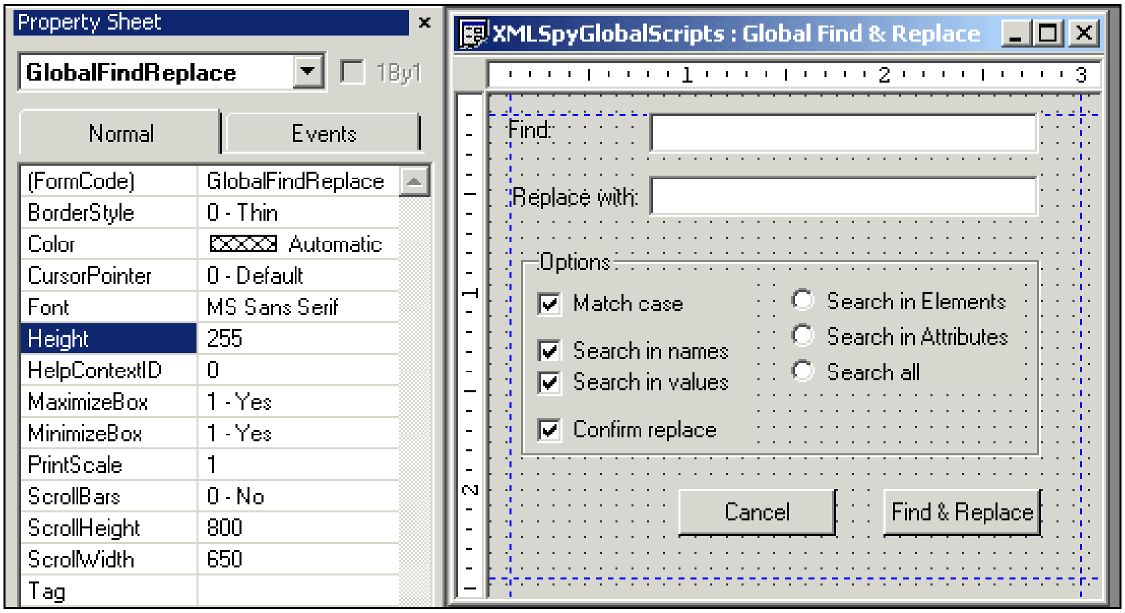


```
function BtnFindReplace_EventClick() {
    This = BtnFindReplace;

    var nIndex = 0;
    strFind = EditFind.Text;

    if(strFind.length > 0)  {
        // close dialog before we do the ac
        strReplace = EditReplace.Text;
        bMatchCase = CheckCase.Value;
        bCheckNames = CheckNames.Value;
        bCheckValues = CheckValues.Value;
        bConfirm = CheckConfirm.Value;
```

Design View
A Form must be created and edited in the editable Design View of the Form. To specify the settings of the workarea, use the menu item **Layout | Grid Settings...**.

In order to insert objects into the Form, use the Form Objects in the Form Object Bar. Build up the visual design of your Form by placing objects from the Form Object Bar at suitable locations within the Form outline. Specify visual refinements of the layout through the **Layout** menu items.

To specify the properties of the Form, either right-click in an empty area of the Form or select **Layout | Properties...**. This displays the Properties Sheet of the Form, where you can define the size of the Form, its font, buttons, and so on.



(FormCode)	GlobalFindReplace
BorderStyle	0 - Thin
Color	▨ Automatic
CursorPointer	0 - Default
Font	MS Sans Serif
Height	255
HelpContextID	0
MaximizeBox	1 - Yes
MinimizeBox	1 - Yes
PrintScale	1
ScrollBars	0 - No
ScrollHeight	800
ScrollWidth	650
Tag	

Find:

Replace with:

Options

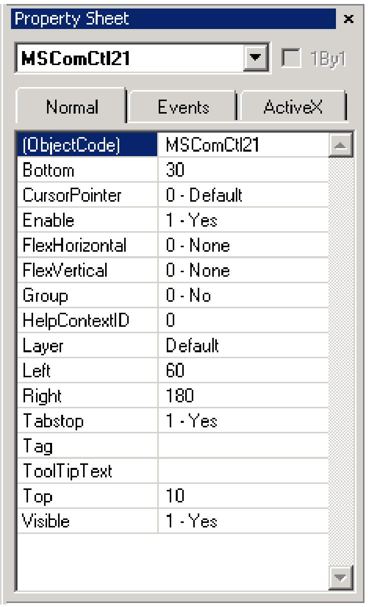
☑ Match case
☑ Search in names
☑ Search in values
☑ Confirm replace
○ Search in Elements
○ Search in Attributes
○ Search all

To switch to the Property Sheet of a particular object, either select that object in the design view or select the name of that object from the drop-down list of the combo box in the Property Sheet.

To close the Property Sheet, click the **Close** button.

The Property Sheet
The Property Sheet lists all properties and events of the selected Form or Form Object when the editable Design View of the Form is active. It enables you to modify properties and to add or modify code for properties and Form event handlers. The combo box at the top of the Property Sheet lists all the properties and events of the active Form.

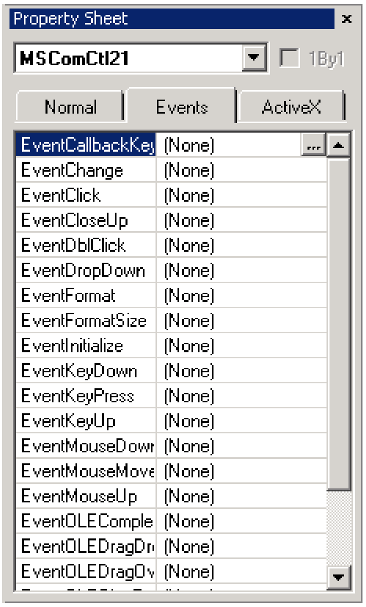
The Property Sheet has three tabs: Normal, Events, and ActiveX. The screenshots below show the three tabs of the Microsoft DatePicker ActiveX control. Note that different Form Objects have different properties and events, and not all objects have three tabs in the Property Sheet.


Property Sheet

MSComCtl21 □ 1By1

Normal | Events | ActiveX

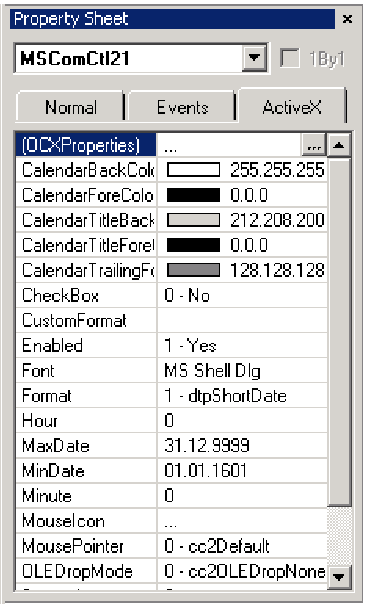
(ObjectCode)	MSComCtl21
Bottom	30
CursorPointer	0 - Default
Enable	1 - Yes
FlexHorizontal	0 - None
FlexVertical	0 - None
Group	0 - No
HelpContextID	0
Layer	Default
Left	60
Right	180
Tabstop	1 - Yes
Tag	
ToolTipText	
Top	10
Visible	1 - Yes


Property Sheet

MSComCtl21 □ 1By1

Normal | Events | ActiveX

EventCallbackKey	(None)
EventChange	(None)
EventClick	(None)
EventCloseUp	(None)
EventDblClick	(None)
EventDropDown	(None)
EventFormat	(None)
EventFormatSize	(None)
EventInitialize	(None)
EventKeyDown	(None)
EventKeyPress	(None)
EventKeyUp	(None)
EventMouseDowr	(None)
EventMouseMove	(None)
EventMouseUp	(None)
EventOLEComple	(None)
EventOLEDragDr	(None)
EventOLEDragOv	(None)


Property Sheet

MSComCtl21 □ 1By1

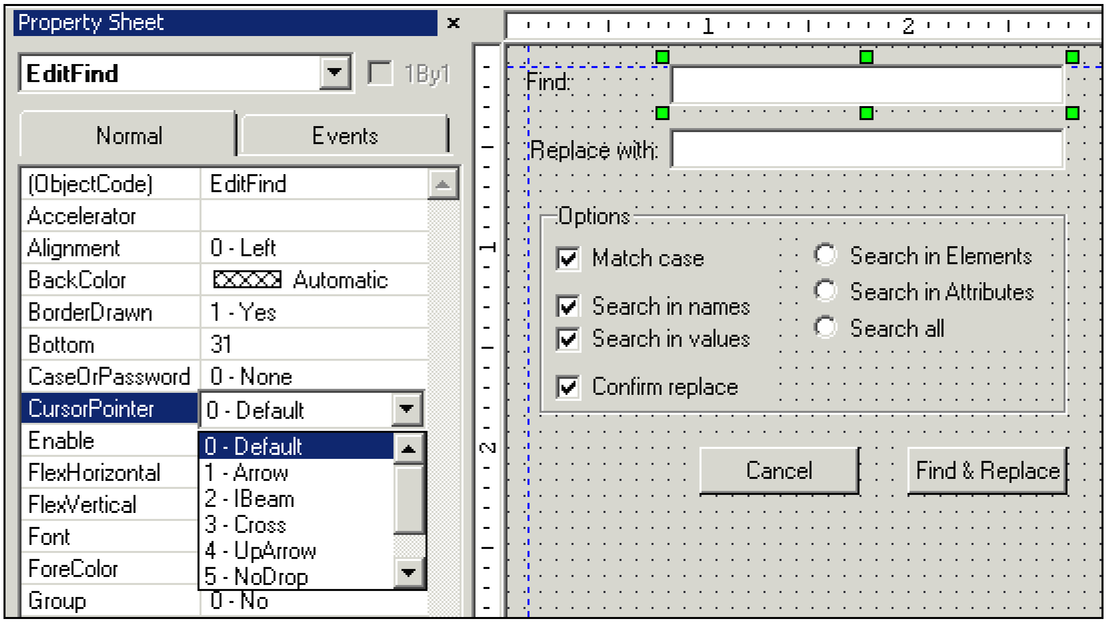
Normal | Events | ActiveX

(OCXProperties)	...
CalendarBackColc	□ 255.255.255
CalendarForeColo	■ 0.0.0
CalendarTitleBack	▨ 212.208.200
CalendarTitleFore	■ 0.0.0
CalendarTrailingFc	▨ 128.128.128
CheckBox	0 - No
CustomFormat	
Enabled	1 - Yes
Font	MS Shell Dlg
Format	1 - dtpShortDate
Hour	0
MaxDate	31.12.9999
MinDate	01.01.1601
Minute	0
MouseIcon	...
MousePointer	0 - cc2Default
OLEDropMode	0 - cc2OLEDropNone

The **Normal tab** specifies the name of the object and properties relating to the object's appearance in the Form. The name is specified by the `ObjectCode` property (here `MScomCtl21`). If you write a script and need access to methods or properties of this object, you must use this name. For example:

```
var strTmp;
strTmp = "";
strTmp = MSComCtl21.Year + '-' + MSComCtl21.Month + '-' +
    MSComCtl21.Day;
```

If a property has enumerated values, then these become available in a drop-down menu when the property is selected (see screenshot below).


Property Sheet

EditFind □ 1By1

Normal | Events

(ObjectCode)	EditFind
Accelerator	
Alignment	0 - Left
BackColor	☒ Automatic
BorderDrawn	1 - Yes
Bottom	31
CaseOrPassword	0 - None
CursorPointer	0 - Default
Enable	0 - Default 1 - Arrow 2 - IBeam 3 - Cross 4 - UpArrow 5 - NoDrop
FlexHorizontal	
FlexVertical	
Font	
ForeColor	
Group	0 - No

Find:

Replace with:

Options

☑ Match case
☑ Search in names
☑ Search in values
☑ Confirm replace

○ Search in Elements
○ Search in Attributes
○ Search all

Cancel | Find & Replace

In the above screenshot, the `EditFind` object is selected. When the `CursorPointer` property is selected, a drop-down menu with the possible values for this property becomes available. This enables you to quickly and correctly select a suitable value for the property.

In the **Events and ActiveX tabs**, to enter code for a property, first select the property for which you want to enter code. A button with an ellipsis (`...`) appears at the right of the entry field. Click this button to open a Script Editor window for that code fragment.

Using layers in Form design

When you insert Form Objects while in the Design View of the Form, these objects are created by default on a layer titled "Default". You can add additional layers **below the Default layer** if you wish to create a Form with backgrounds objects. The objects in each layer are superimposed on the objects of underlying layers if there is an overlap. Layers below the Default layer can be moved relative to each other (i.e. up or down), but the Default layer remains the top layer. As a result, objects in the Default layer will be superimposed on all objects in lower layers. For details of using layers, see Layout.

Setting the tab sequence for data-input objects

When users input data in a Form, they commonly move from one input field to the next by pressing the Tab key. You can specify the tab sequence in the Objects Sheet (Tab Order) dialog. See Layout for details.

Also see:

- The Layout menu for commands used to layout a Form. There are a range of commands that help you to format
- Creating a Form for more details about creating a Form.

2.3 Using the Scripting Environment

The Forms, Event Handlers, and Macros you create in the Scripting Environment use scripts that interact with the XMLSpy API, which is a COM API. The variables and functions required for this interaction are defined in the **Global Declarations** module of the Scripting Environment. The object model of the XMLSpy API is documented in the XMLSpyAPI documentation.

Event Handler scripts can be written for standard XMLSpy events as well as for custom events. This allows you to add and modify event handling in your XMLSpy interface. See Creating an Event-handler for an example of how to work with Event Handlers in XMLSpy.

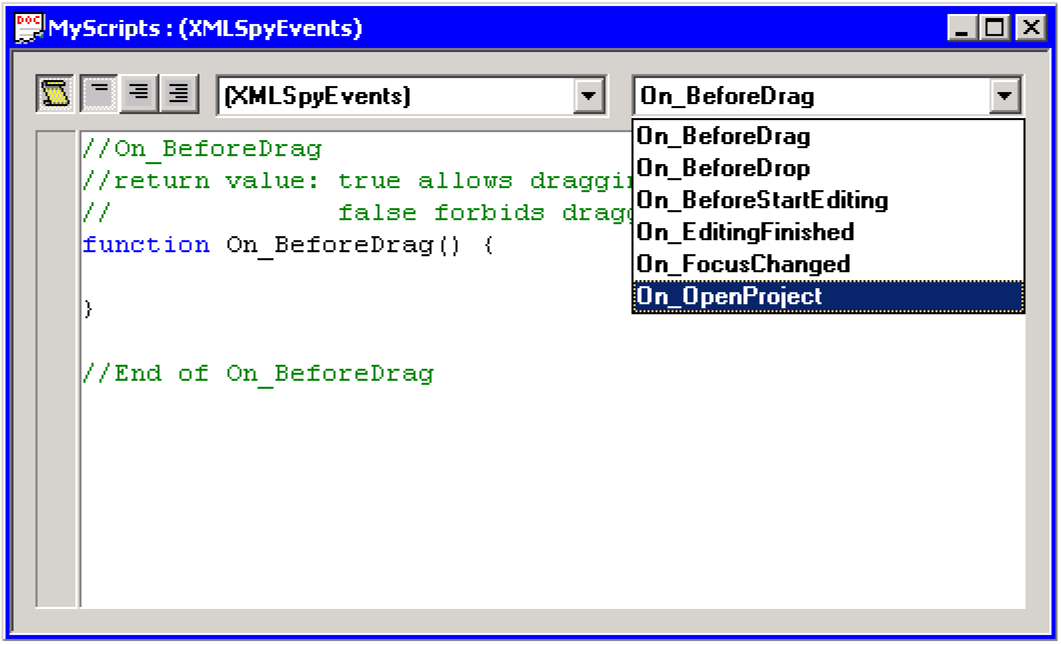
When you create a **Form**, you require variables to accept user input and functions that carry out the required functionality. Both variables and functions are defined in the Global Declarations. In addition, you need to reference some event handler to execute the Form in the desired way. Creating a Form describes the steps required to create a Form.

Macros automate repetitive or complex tasks. In the Scripting Environment, you can create a script that calls functions both XMLSpy functions as well as custom functions you define in the Global Declarations module. This flexibility provides you with powerful method of automating tasks within the XMLSpy IDE. How you create a Macro is described in Writing a Macro.

The subsections of this section explain, with examples, how each of the above is to be created. The section ends with a subsection explaining how Macros are called in XMLSpy.

2.3.1 Creating an Event Handler

To create an Event Handler for application events (as opposed to Form events), double-click the XMLSpyEvents Module in the Project Window. This pops up a scripting window for event handlers in the Main Window.



```
//On_BeforeDrag
//return value: true allows draggi
//              false forbids drag
function On_BeforeDrag() {

}

//End of On_BeforeDrag
```

The combo box on the right lists all the application events. To access a particular event-handler script, click the relevant event in the righ-hand combo box. This pops up the script which you can edit or a script outline you can fill in. If the event you want to write code for does not have an entry in the Event combo box, use the **Project | Add Function** command (or right-click in the script window and select **Add Function**) to add an Event.

Note:

- Event Handlers need function headers with the correct spelling of the event name. Otherwise the Event Handler will not get 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();
  }
  ```

- While a Macro is executed, Event Handlers from XMLSpy are not processed.
- The best way to re-create a deleted event handler is to create a new scripting project and copy name and function definition from there.
- In order for events to be processed, the Process Events options must be toggled on in the Settings dialog of XMLSpy. See Scripting Settings for details.
- Also see Programming points to note.

Example

The screenshot above shows the XMLSpyEvents module with a list of predefined events. The `On_OpenProject` allows you to add a script that will be executed each time 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_OpenProject()` event, and then save the scripting project.

```
function On_OpenProject()
{
 var bOK;
 var nIndex,nCount;
 var objItems,objXMLFolder = null;

 objItems = Application.CurrentProject.RootItems;
 nCount = objItems.Count;

 // search for XML folder
 for(nIndex = 0;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;
```

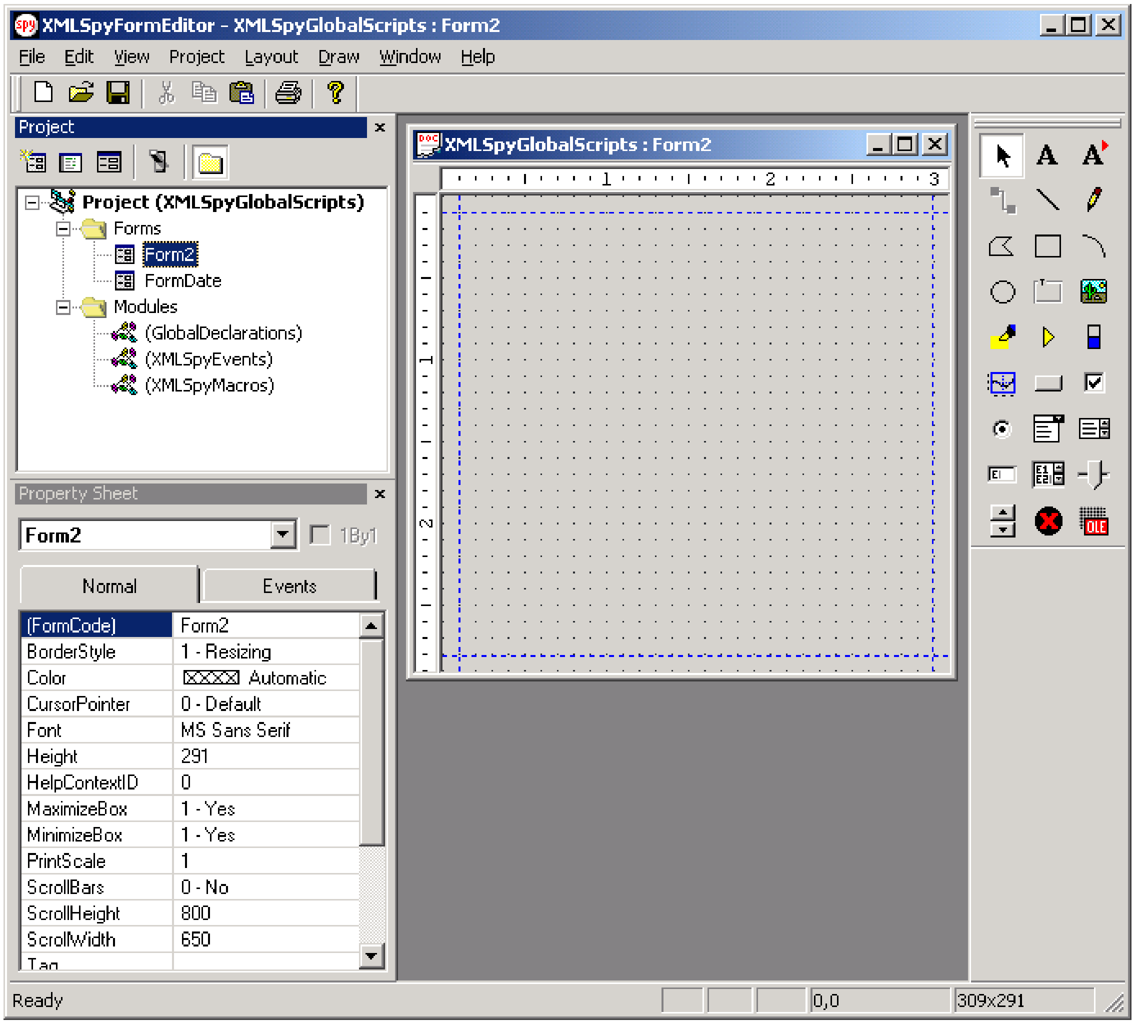
```
 // step through associated xml files
 for(nIndex = 1;nIndex <= nCount;nIndex++) {
  objChild = objXMLFolder.ChildItems.Item(nIndex);

  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 Open Project event is handled.

2.3.2 Creating a Form

A Form queries users for input (data and/or a choice), and executes the required action based on the input. In the Scripting Environment, you can graphically design a Form as well as create the scripts to perform the required action. The example below takes you through the steps needed to create a Form. In it we create a Form in which the user can type in the name of elements that are to be deleted from the active XML file in XMLSpy.

Creating a new Form
To add a new Form to a scripting project, select **Project | Add Form** or click the **New Form** button in the toolbar of the Project Window. The new Form is added immediately. It appears as a separate window (see screenshot below), and its title appears in the Project Window. Open the Property Sheet for the Form by either right-clicking within the Form Window or selecting **Layout | Properties...**.


XMLSpyFormEditor - XMLSpyGlobalScripts : Form2
File Edit View Project Layout Draw Window Help
Project
Project (XMLSpyGlobalScripts)
Forms
Form2
FormDate
Modules
(GlobalDeclarations)
(XMLSpyEvents)
(XMLSpyMacros)
XMLSpyGlobalScripts : Form2
Property Sheet
Form2
1By1
Normal
Events
Ready
0,0
309x291

(FormCode)	Form2
BorderStyle	1 - Resizing
Color	☒☒☒☒ Automatic
CursorPointer	0 - Default
Font	MS Sans Serif
Height	291
HelpContextID	0
MaximizeBox	1 - Yes
MinimizeBox	1 - Yes
PrintScale	1
ScrollBars	0 - No
ScrollHeight	800
ScrollWidth	650
Tag	

Naming the Form

You name a Form by entering a name for it in the line `FormCode` of the Property Sheet . For our example, enter `RemoveDlg` as the name of the Form. After entering the name, press Enter. The name of the Form now also appears in the title box of the Propery ySheet.

Inserting a Form Object

We need an edit box in which the user can type the names of the elements to be removed from the XML file. Select the edit box icon from the Form Objects Bar on the right side of the Main Window. Then, in the Design View of the `RemoveDlg` Form, draw, with the mouse, a rectangle for the edit box. This rectangle sets out the position and dimensions of the edit box. The edit box is created in the Design View, and (if **Layout | Properties...** is toggled on) the Property Sheet for the edit box is displayed. (To display the Property Sheet for the edit box, either select the edit box and toggle on **Layout | Properties...** or right-click the edit box.)

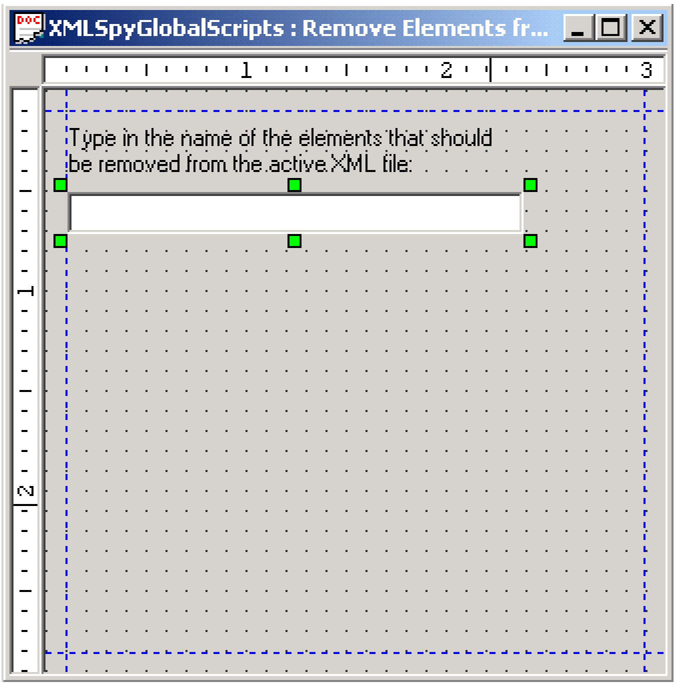
In the `(ObjectCode)` line of the Property Sheet, type in `EditElements` as the name of the edit box.

Position and dimension of a Form Object

The position and dimensions of the edit box are specified in the Property Sheet in terms of the Left, Right, Top, and Bottom properties. The Left and Right properties take values that are the distances (in pixels) of the left and right borders of the edit box, respectively, from the left border of the Form. The Top and Bottom properties take values that are the distances (in pixels) of the top and bottom borders of the edit box, respectively, from the top border of the Form.

Inserting text in the Form

Add a caption for the edit box with the static text icon A from the Form Objects Bar. The Design View of the Form should look something like this:



Type in the name of the elements that should be removed from the active XML file:

Inserting buttons

We now need two buttons in our form: **Delete** and **Cancel**. To insert these, do the following:

1. Select the button icon ▭ and then draw the two buttons in the Design View.
2. Name each button in the `(ObjectCode)` line of the respective Property Sheets: `BtnDelete` and `BtnCancel`.
3. Specify the Button text (**Delete** and **Cancel**, respectively) either by editing the button text directly in the Design View or by editing the value of the `Text` property in the Property Sheet.

Writing the required scripts

After the visual design of the Form is complete, we need to associate the Form Objects with suitable scripts. In this case, we need two types of scripts:

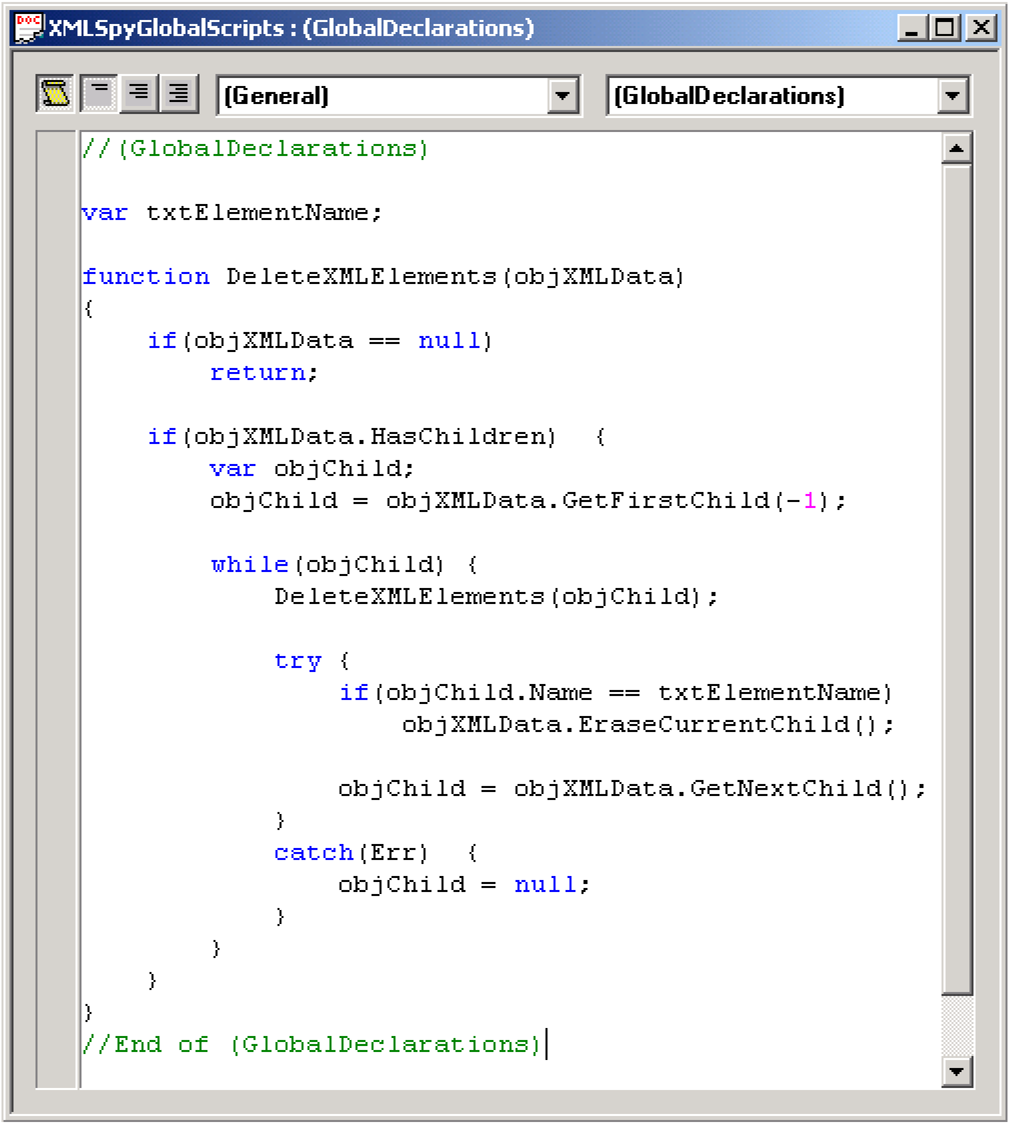
- A function that loops through the XML file, deleting all `XMLData` objects with a given name (specified by the user in the edit box)
- Event handlers that are executed when the two buttons are clicked

Creating the function

The function that deletes the `XMLData` objects must be defined in the Global Declarations module of the scripting project. Open the Global Declarations window by double-clicking the `GlobalDeclarations` module in the Project Window. Type the following code into the Global Declarations window:

```
var txtElementName;

 function DeleteXMLElements(objXMLData)
 {
   if(objXMLData == null)
     return;

   if(objXMLData.HasChildren){
     var objChild;
     objChild = objXMLData.GetFirstChild(-1);

     while(objChild) {
      DeleteXMLElements(objChild);

      try{
        if(objChild.Name == txtElementName)
        objXMLData.EraseCurrentChild();

        objChild = objXMLData.GetNextChild();
      }
      catch(Err) {
        objChild = null;
      }
    }
  }
}
```

The Global Declarations window should look like this:



```
//(GlobalDeclarations)

var txtElementName;

function DeleteXMLElements(objXMLData)
{
    if(objXMLData == null)
        return;

    if(objXMLData.HasChildren)  {
        var objChild;
        objChild = objXMLData.GetFirstChild(-1);

        while(objChild) {
            DeleteXMLElements(objChild);

            try {
                if(objChild.Name == txtElementName)
                    objXMLData.EraseCurrentChild();

                objChild = objXMLData.GetNextChild();
            }
            catch(Err)  {
                objChild = null;
            }
        }
    }
}
//End of (GlobalDeclarations)
```

Creating the button events

- Select the `BtnDelete` button and click the **Events** tab in the Property Sheet.
- Select the `EventClick` property and click the ellipsis button ... at the right. A Script Editor window pops up with the predefined function header of the click-event. This function will be executed every time the user clicks the **Delete** button. Add the following code to the event:

```
if(EditElements.Text != "")      {
             txtElementName = EditElements.Text;
             DeleteXMLElements(Application.ActiveDocument.RootElement);
}
```

Summary of Form creation

The Form we have created using the steps above does the following:

- The **Delete** button sets the global variable `txtElementName` to the string that the user enters in the edit box (`EditElements` is the name we have assigned the edit box).
- The global variable `txtElementName` calls the function `DeleteXMLElements()` with

the root element of the active XML document. (The function `DeleteXMLElements()` is the one we have defined in the Global Declarations module.)

- The `DeleteXMLElements()` function uses the `XMLData` object of the XMLSpy API.

Running the Form
To run the Form, create a Macro as follows:

1. Open the Macros module (by double-clicking `XMLSpyMacros` in the Project Window).
2. Create a new Macro called `DeleteElements` and enter thefollowing code:

```
var a;
if(Application.ActiveDocument != null)
              a = Application.ShowForm("RemoveDlg");
```

3. Run the Macro by selecting the `DeleteElements` in the Show Macros dialog, which you open by clicking **Tools | Show macros...** in the XMLSpy GUI.

Note:

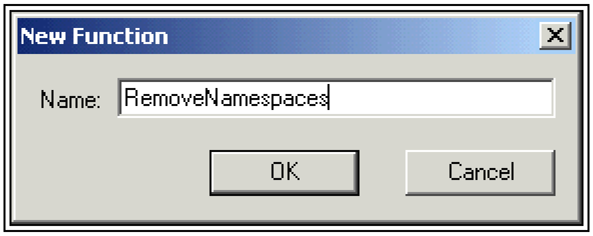
- A Form can display another Form with the function `Application.ShowForm()`.

2.3.3 Writing a Macro

A Macro automates a repetitive or complex task. In this section, we write a Macro that removes all namespace prefixes from the active document.

Creating a new Macro

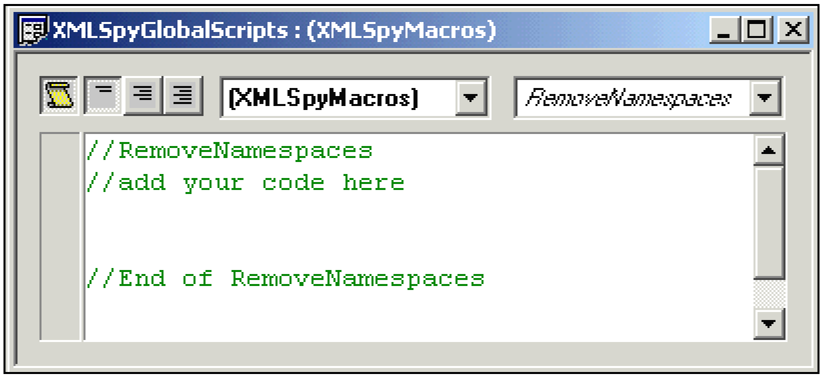
1. To create a new Macro, right-click `XMLSpyMacros` in the Project Window and select **Add Function** from the context menu.
2. In the New Function dialog that appears, enter `RemoveNamespaces` for the name of the Macro.



Name: RemoveNamespaces

The Scripting Environment creates a new Macro with this name and adds it to the `XMLSpyMacros` module.

3. Double-click the `XMLSpyMacros` module in the Project Window to open the Macros window.
4. Select the `RemoveNamespaces` Macro.



```
//RemoveNamespaces
//add your code here


//End of RemoveNamespaces
```

5. Enter the code below in the Macro window:

```
if(Application.ActiveDocument != null)  {

RemoveAllNamespaces(Application.ActiveDocument.RootElement);
             Application.ActiveDocument.UpdateViews();
}
```

Note: Macros do not support parameters or return values, so no function header should exist in the macro implementation.

Creating the function used in the Macro

We now have to write the `RemoveNamespaces` function. Creating it in the Global Declarations module makes it accessible to all Macros, Event Handlers, and Forms.

1. Activate the Global Declarations module by double-clicking it in the Project Window.
2. Append the following code for the `RemoveNamespaces()` function to code for any previously defined global variables or functions:

```
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;
          }
       }
    }
  }
```

This completes the creation of the new macro.

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. The function `DeleteXMLElements()` in Creating a Form, for example, calls itself recursively.

Running the Macro

Run the Macro by selecting the `DeleteElements` in the Show Macros dialog, which you open by clicking **Tools | Show macros...** in the XMLSpy GUI. See Calling macros.

Note: While a Macro is executed, Event Handlers from XMLSpy are not processed.

Modifying the Macro

This Macro can be easily modified to rename namespaces rather than remove them. You would need to carry out the following steps:

1. Design a Form (see Creating a Form) in which the user can specify the old and the new namespace names.
2. In the Macro code, change the `try-catch` block of the `RemoveAllNamespaces()` function to something like this:

```
try{
 var nPos,txtName;
 txtName = objChild.Name;

 if((nPos = txtName.indexOf(":")) >= 0){
  var txtOld;
  txtOld = txtName.substring(0,nPos);

  if(txtOld == txtOldNamespace)
   objChild.Name = txtNewNamespace + ":" + txtName.substring(nPos+1);
 }

 objChild = objXMLData.GetNextChild();
}
catch(Err) {
 objChild = null;
}
```

This code assumes that `txtOldNamespace` and `txtNewNamespace` are declared as global variables and are set with the proper values.

2.3.4 Calling macros

There are two ways to call or run a Macro:

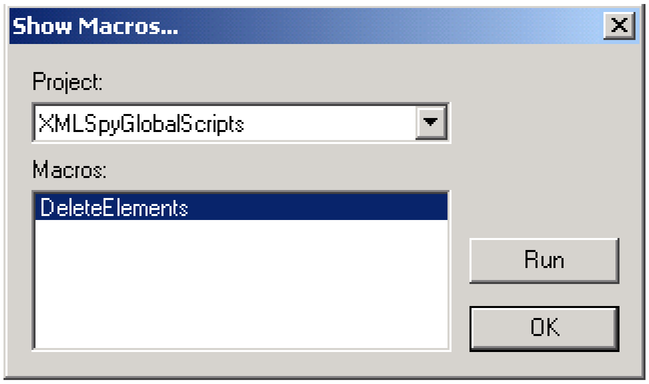
- With the **Show Macros...** dialog of XMLSpy
- By creating and using a menu item (or Run command) for the Macro **in the Tools menu** of XMLSpy
- By creating and using a toolbar button for a macro

Note: While a Macro is executed, Event Handlers from XMLSpy are not processed.

Using the Show Macros... dialog

1. In XMLSpy, click **Tools | Show macros...**. This pops up the Show Macros... dialog, which displays all macros defined in the global scripting project and in the scripting project assigned to the current XMLSpy project. In the Project combo box, select

whether the Macros in the global scripting project or the Macros in the assigned scripting project are to be displayed.



2. Select the Macro you wish to run.
3. Click the **Run** button. Before the Macro is executed, the Show Macros... dialog is closed.

Creating a menu item in the Tools menu

The XMLSpy API includes a function to add Macros as menu items to the **Tools** menu. This function can be used to add one or more Macros to the **Tools | Show macros...** list of Macros. To do this carry out the following steps:

1. To add a menu item, call the XMLSpy API function `AddMacroMenuItem()`.
2. To reset the Tools menu call `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.

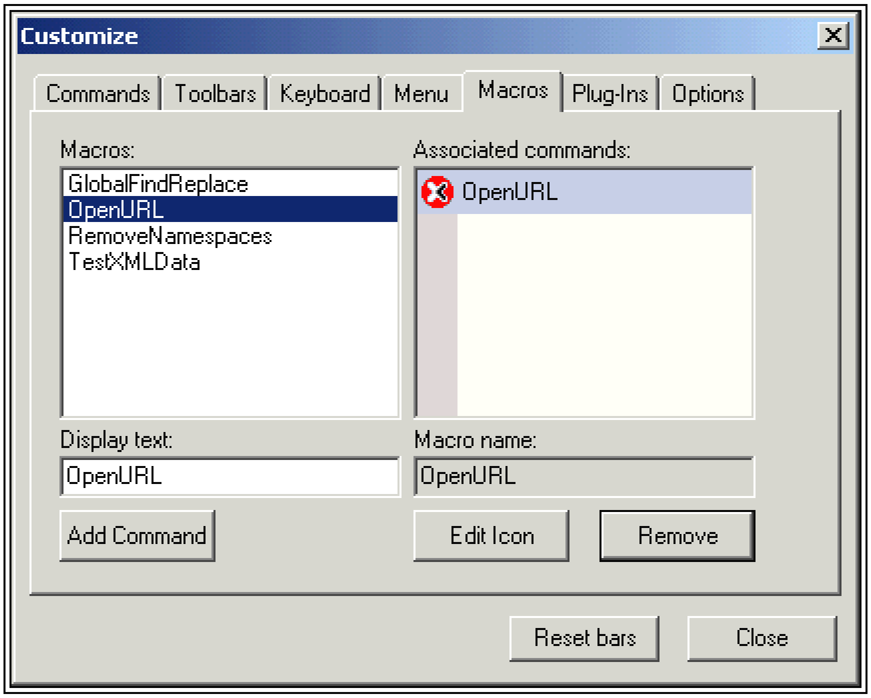
Example:

```
Application.AddMacroMenuItem("DeleteElements","Delete Elements
    Dialog");
```

- The first parameter (`DeleteElements` in the example) 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.

Using a toolbar icon for a macro

You can create an icon in the toolbar, which, when it is clicked, will run a Macro. To do this, click **Tools | Customize | Macros**.



Now do the following:

1. In the Macros pane, select the required Macro.
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 when the cursor changes from an arrow to an I-beam.
6. Click Reset bars for this Macro icon and command to be displayed in the **Tools | Show Macros...** dialog.

2.3.5 Programming points to note

The following programming points should be noted:

- Out-parameters from methods of the XMLSpy API require special variables in JavaScript. Given below are some examples.

```
// use JavaScript 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);
```

2.4 Menus

2.4.1 File

New Project
Creates a new project. You are prompted for the scripting language of the new project, which can be either JavaScript or VBScript. The project is opened as the active project, and its title is diaplayed in bold in the Projects Window.

Open Project
Opens an existing project. The newly opened project becomes the active project. Only one project among all the open projects can be active at a time. A project is made the active project automatically when it is opened or if it is created as a new project. Otherwise, a project is made active by explicitly specifying this—right-click a project title and select **Set as Active Project**.

If only one instance of the project is open, then its title is displayed in bold. Multiple instances of a single project can be open, and in this case the title of the currently active instance or of the last instance to have been active is displayed in bold.

Close Project
Closes the selected project. If a project is not the active project, closing it automatically makes it the active project before it is closed—therefore, after it is closed no project is active.

Save Project
Saves the selected project.

Save Project As
Saves an unnamed open project with a name or a named open project under another name.

Printing
You can print a form or script by using the Print command. A Print Preview is also available. Note that to switch back from the Print Preview to the Scripting Environment, you should press the **Close** button. Do not close the Print Preview window; this will cause the entire Scripting Environment to close.

Switch to XMLSPY...
Minimizes the Scripting Environment and restores the XMLSpy window.

Exit
Closes the Scripting Environment window.

2.4.2 Edit

Undo and Redo
The Undo and Redo commands enable you to carry out multiple undo and redo commands.

Cut, Copy, Paste, Delete
To cut, copy and delete, select an object or text fragment, and select one of these commands. To paste, place the cursor at the desired location and select the Paste command.

2.4.3 View

Toolbars
Toggles on/off a number of toolbars.

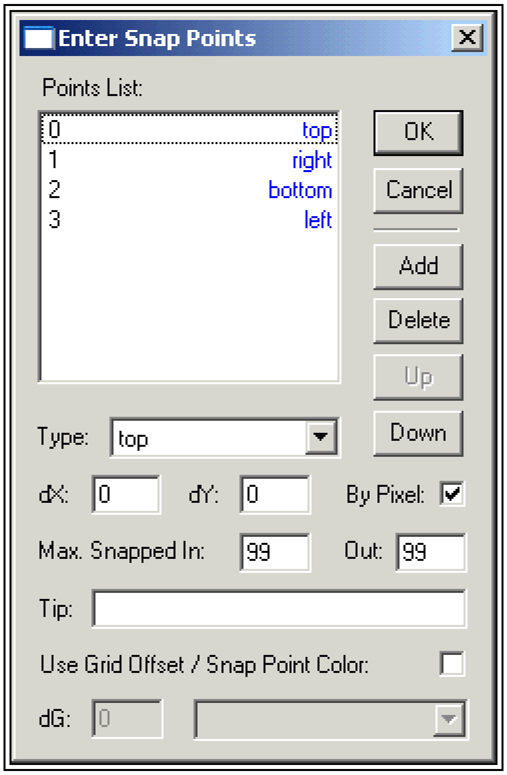
Project Bar

Toggles on/off the Project Bar.

Status Bar
Toggles on/off the Status Bar at the bottom of the Sripting Environment window.

Ruler Bars
Toggles on/off the ruler guides along the top and left margins of the page.

Snap Points
Toggles on/off the snap points of an object. Snap points are the points on an object that will snap to another object when that object is being moved. You can add and delete snap points by selecting the object, and, in the Property Sheet of the object, clicking the ellipsis in the value field of the AnchorSnaps property. This pops up the Enter Snap Ponts dialog, in which you can add snap points.



Points List:

0	top
1	right
2	bottom
3	left

Type: top

dX: 0 dY: 0 By Pixel: ☑

Max. Snapped In: 99 Out: 99

Tip:

Use Grid Offset / Snap Point Color: □

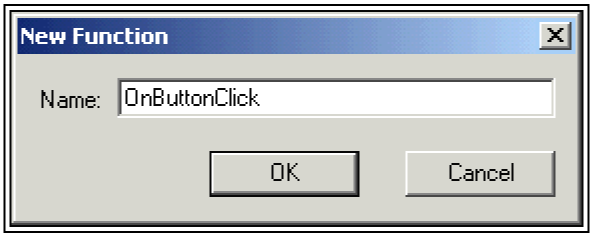
dG: 0

When the snap points are toggled on, they are visible and aid you to position objects accurately.

2.4.4 Project

Add Form
Enables you to add a Form to the current set of Forms in the project.

Add Function
When Events or Macros is selcted in the Project Window, you can add a Function to the list of already available functions. Selecting this command pops up the following dialog:



OnButtonClick

Entering the name of the function and clicking OK adds a function with this name to the list of existing functions and opens a scriptig window for this function.

Remove Function
This command opens the Remove Function dialog, in which you enter the name of the Events or Macros functon to be removed.

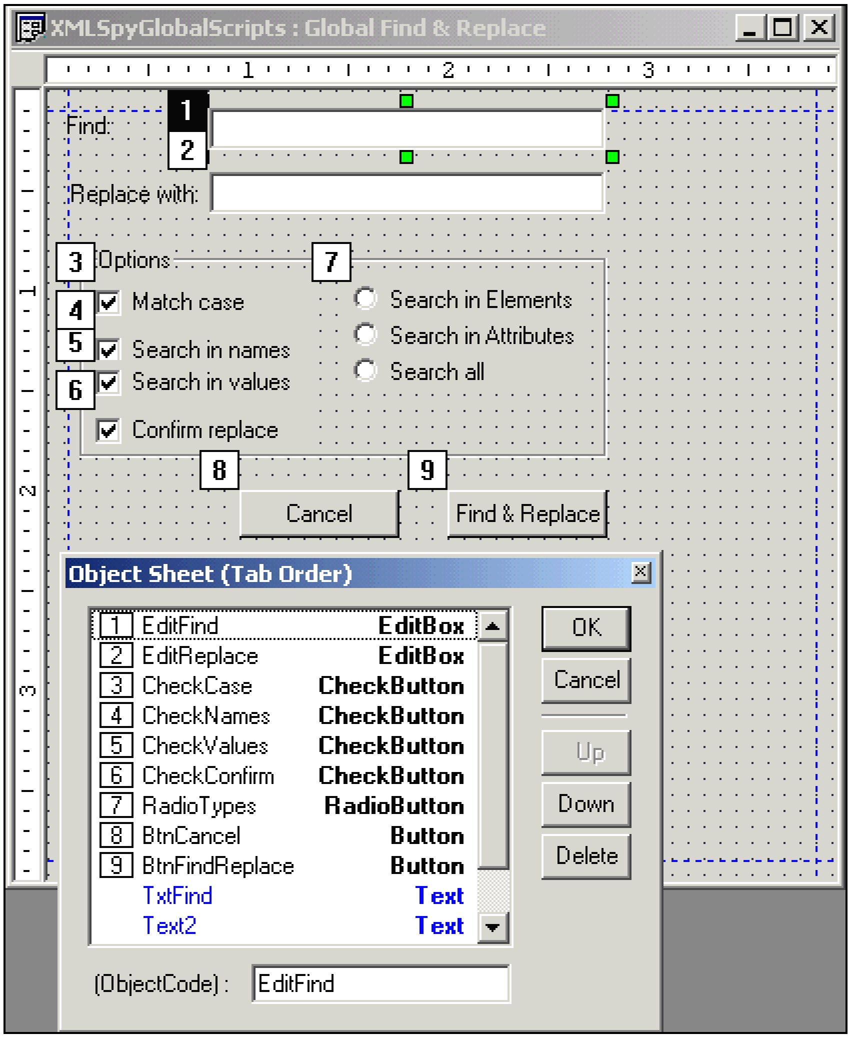
Project Information
This command gives you the path to the current project file and information about whether the project was opened from within XMLSpy or not.

2.4.5 Layout

The Layout menu contains commands for Form layout and view controls. These commands are not relevant for Modules.

Spacing, positioning, and sizing of objects
When two or more Form Objects are selected (by pressing Shift while clicking the objects), they can be spaced and sized relative to each other as well as positioned relative to the borders of the Form. You can do this by using the **Align Objects**, **Space Evenly**, **Align in View**, and **Make Same Size** commands. Note that to use the **Space Evenly** command at least three objects have to be selected.

Objects
The **Objects...** command pops up the Object Sheet (Tab Order) window, in which you specify the tab sequence of the data-input fields. Data-input fields correspond to data-input Form Objects.



☑ Match case

☑ Search in names

☑ Search in values

☑ Confirm replace

○ Search in Elements

○ Search in Attributes

○ Search all

1	EditFind	EditBox
2	EditReplace	EditBox
3	CheckCase	CheckButton
4	CheckNames	CheckButton
5	CheckValues	CheckButton
6	CheckConfirm	CheckButton
7	RadioTypes	RadioButton
8	BtnCancel	Button
9	BtnFindReplace	Button
	TxtFind	Text
	Text2	Text

(ObjectCode) : EditFind

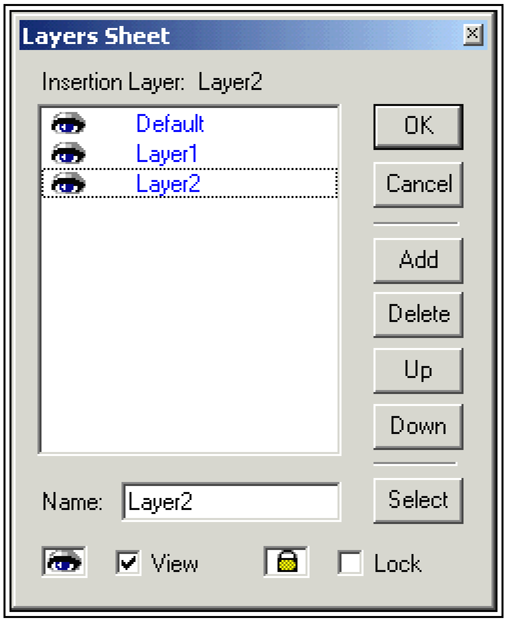
When you insert a Form Object in your design, the new object is appended to the appropriate section (numbered or non-numbered) in the Object Sheet list. The Object Sheet list has two parts, a numbered part for data-input objects and a non-numbered part for display objects. The number of an object in the numbered part indicates the position of that object in the run-time tab sequence. You can change the position of an object in the tab sequence by selecting it and pressing the **Up** or **Down** buttons as required. You can also delete an object by pressing the **Delete** button.

Note: The tab sequence applies to data-input objects only at run-time, i.e. when the Form is displayed for user input. In the Form Editor, pressing the Tab key first takes you through the display objects (in the order in which these are listed in the Object Sheet), and then through the data-input objects (in the tab sequence).

Layers

A Form is designed in a "Default" layer that is always the top layer. You can add additional layers below the Default layer, and move these lower layers relative to each other. The Default layer remains the top layer. Layers, therefore, allow you to add background effects to your Form.

To add and manipulate layers, select the **Layers...** command. This pops up the Layers Sheet.



Insertion Layer: Layer2

Name: Layer2

☑ View □ Lock

To add a new layer, click the **Add** button. You can rename a layer by selecting it and typing a new name into the **Name** input field. To delete a layer, select the layer to be deleted and click the **Delete** button. To move a layer up or down in the layer sequence, select the layer to move and press the **Up** or **Down** buttons as required. The layer sequence is ordered with the first layer in the Layers Sheet corresponding to the topmost layer of the Form.

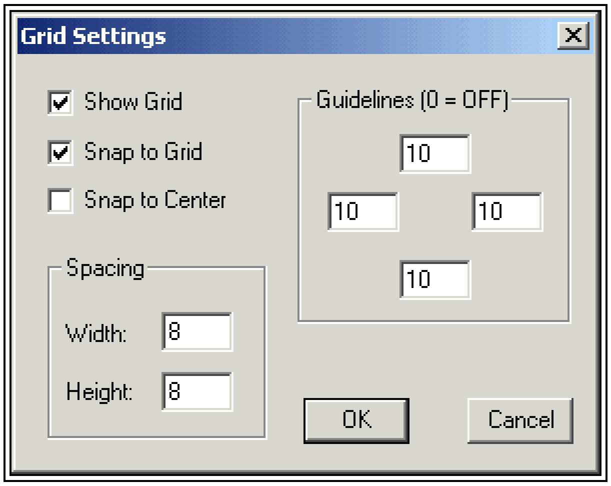
The **Select** button selects, in the design view, all the objects of the currently selected layer. You can also turn the display of all objects in a layer on or off by using the **View** check box. Locking a layer, with the **Lock** check box, renders that layer uneditable and causes the Object Bar to be grayed out when that layer is active.

Properties

The **Properties...** command pops up the Property Sheet for the selected Form. A Property Sheet is available not only for each Form but also for each Form Object. You can modify properties of a Form or Form Object via its Property Sheet. How to use Property Sheets is described in detail in The Form Editor.

Grid Settings

The **Grid Settings...** command pops up the Grid Settings dialog, in which you can specify the appearance of the background grid for the design view.



Grid Settings

☑ Show Grid

☑ Snap to Grid

☐ Snap to Center

Spacing

Width: 8

Height: 8

Guidelines (0 = OFF)

10

10 10

10

Spacing settings specify the horizontal and vertical distances between grid lines. Four guidelines are available (top, right, bottom, left), and you can specify the offset of each from the corresponding border of the Form. Grid Settings take effect with the next design view to be made active.

Edit View

This is a toggle setting that is specified individually for each Form. Checking the **Edit View** command causes the Form to be opened in editable design view or for it to switch from non-editable view to editable design view. Unchecking the **Edit View** command causes the Form to be opened in non-editable view or for it to switch from editable design view to non-editable view. Note that the setting is made for the active Form only; it does not apply to all Forms.

Edit Script

This is a toggle setting that is specified individually for each Form. Cecking the **Edit Script** command causes a separate script window to be displayed for that Form. Unchecking the command causes the script window to close.

2.4.6 Draw

Order

This is a Form Editor command and enables you to place the selected object in front of or behind other objects on the same layer. If you wish to arrange objects which are on different layers, then you should manipulate layers in the Layers Sheet (accessed with the **Layout | Layers** command).

Zoom

This is a Form Editor command menu and enables you to zoom in and out on an object or the Form as a whole.

Group

This is a Form Editor command menu which is enabled when two or more objects, or a group of objects, are selected. It allows you to group and ungroup two or more objects. This is useful if you wish to keep objects together as a group.

Rotate

Enables you to rotate certain objects. The **Free Rotate** command is a toggle command. It allows you to grab the object by one of its handles and rotate it about its center. You acn also **Rotate Right** by 90º. The **Unrotate** command functions like an Undo command.

2.4.7 Window

The Window menu enables you to manipulate the arrangement of open windows in the Main Window. You can cascade and tile windows, and arrange icons. You can also use this menu to select, from a list of all open windows, the window you wish to make the active window.

3 XMLSpy 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++. See the folder "xmlspy\Examples\XMLSpyPlugIn" of your XMLSpy installation for a sample using VisualBasic.

Note: If you are interested in developing a Plugin for XMLSPY using this API, please visit the Altova Partner Portal at **http://partners.altova.com**, or contact us via e-mail at mailto:partners@altova.com to learn more about the different partnership options!

3.1 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.

3.2 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.

3.3 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

3.4 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.

3.4.1 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-A1F3-270692EO8AFC),

//   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 -----
```

```
 };
};
```

3.4.2 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 _stdcall OnCommand(long nID, IDispatch* pXMLSpy);

 virtual HRESULT _stdcall OnUpdateCommand(long nID, IDispatch* pXMLSpy,
SPYUpdateAction* pAction);

 virtual HRESULT _stdcall OnEvent(long nEventID, SAFEARRAY **arrayParameters,
IDispatch* pXMLSpy, VARIANT* pReturnValue);

 virtual HRESULT _stdcall GetUIModifications(BSTR* pModificationsXML);

 virtual HRESULT _stdcall GetDescription(BSTR* pDescription);
};

#endif //__XMLSPYIDEPLUGIN_H_
```

3.4.3 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);
}
```

3.5 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.

3.5.1 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
                    objDoc.Save
                    Set objDoc = Nothing
                End If
            End If
        End If

        Set objSpy = Nothing
    End If
End Sub
```

3.5.2 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:

```
       spyEnable        = 1
       spyDisable       = 2
       spyCheck         = 4
       spyUncheck       = 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
```

3.5.3 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:

On_BeforeStartEditing	= 1
On_EditingFinished	= 2
On_FocusChanged	= 3
On_Beforedrag	= 4
On_BeforeDrop	= 5
On_OpenProject	= 6
On_OpenDocument	= 7
On_CloseDocument	= 8
On_SaveDocument	= 9

Events available since XMLSpy 4r4:

On_DocEditDragOver	= 10
On_DocEditDrop	= 11
On_DocEditKeyDown	= 12
On_DocEditKeyUp	= 13
On_DocEditKeyPressed	= 14
On_DocEditMouseMove	= 15
On_DocEditButtonUp	= 16
On_DocEditButtonDown	= 17
On_DocEditContextMenu	= 18
On_DocEditPaste	= 19
On_DocEditCut	= 20
On_DocEditCopy	= 21
On_DocEditClear	= 22
On_DocEditSelectionChanged	= 23

Events available since XMLSpy 2004:

On_DocEditDragOver	= 10

Events available since XMLSpy 2004r4 (type library version 1.4):

On_BeforeOpen Project	= 25
On_BeforeOpenDocument	= 26
On_BeforeSaveDocument	= 27
On_BeforeCloseDocument	= 28
On_ViewActivation	= 29
On_DocEditKeyboardEvent	= 30
On_DocEditMouseEvent	= 31

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.

3.5.4 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
```

3.5.5 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
```

4 The XMLSpy API

The COM-based XMLSpy API of XMLSpy enables other applications to use the functionality of XMLSpy. As a result, it is now possible to automate a wide range of tasks, from validating an XML file to modifying complex XML content (with the XMLData interface). The XMLSpy API is also used by the built-in Scripting Environment of XMLSpy to access application functionality (for a complete description of which, see Scripting). It is also used by IDE Plugins.

XMLSpy and the XMLSpy API follow the common specifications for automation servers set out by Microsoft. It is possible to access the methods and properties of the XMLSpy API from common development environments, such as those using C, C++, VisualBasic, and Delphi, and with scripting languages like JavaScript and VBScript.

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.
- Terminate XMLSpy with the Application.Quit method.
- Don't forget to disable dialogs if the user interface is not visible.
- See Error handling for details of how to avoid annoying error messages.
- Free references explicitly if you are using C or C++.

This documentation
The documentation about the XMLSpy API is broadly divided into two parts.

- The first part consists of an Overview, which describes the object model for the API, important concepts, and Usage Examples to help you get started.
- The second part is a reference section that contains descriptions of all the interface objects of the XMLSpy API.

There have been improvements and updates made from release to release. The differences between releases are documented in Release Notes.

4.1 Overview

This overview of the XMLSpy API provides you with the object model for the API, and with a description of the most important API processes. The following topics are covered:

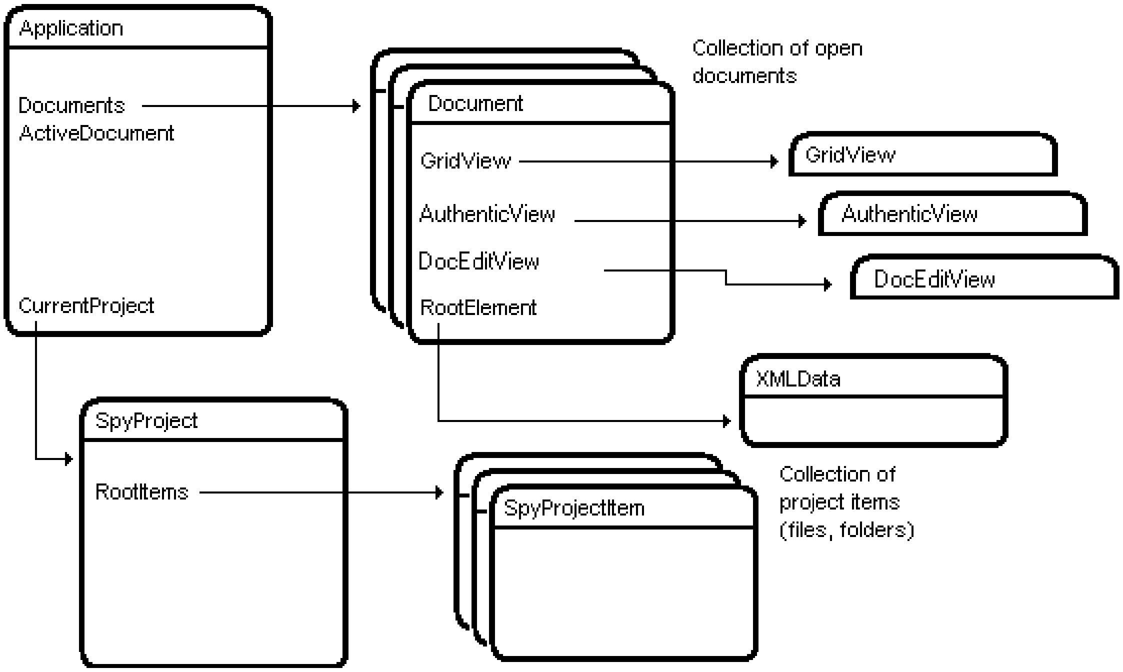
- The object model
- Simple document access
- Error handling
- Events
- Import and export of data
- Using XMLData to modify document structure
- The DOM and XMLData

4.1.1 Object model

The starting point for every application which uses the XMLSpy 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, to use the complete XMLSpy 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 you the links between the main objects of the XMLSpy 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 of 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. Most of the times you either open a project and access the documents from there or you directly open a document via the `Documents` interface.

4.1.2 Simple document access

Create and open files

Use the methods of the `Documents` interface to create and open documents. This interface is accessible via the `Application.Documents` property.

Example:

```
Dim objDoc As Document
Set objDoc = objSpy.Documents.OpenFile("C:\xmlfiles\myfile.xml", False)
```

Sometimes it is necessary to show a file dialog to the user to select a file. This usually happens in the scripting environment and you only need to pass True as the second parameter to `OpenFile()`.

To create a new file use `Documents.NewFile()`. Any existing file with the same path will be overwritten during the next `Document.Save()` call.

Documents collection

All open documents are accessible via the `Documents` collection. All collection objects in the XMLSpy API conform to the Microsoft specifications. So it is possible to use the For-Each loop in VisualBasic to iterate through the open documents.

Example:

```
Dim objDocs As Documents
Dim objDoc As Document

Set objDocs = objSpy.Documents

For Each objDoc In objDocs
 'do something useful with your document
 objDoc.AssignXSL "C:\myXSL.xsl", False
Next
```

Another way to access a document is the `Documents.Item` method. The method takes an index as a parameter and gets the corresponding document object from the collection. Please note that 1 is the first index value. `Documents.Count` retrieves the total number of documents.

Example:

```
Dim objDocs As Documents
Dim objDoc As Document

Set objDoc = objDocs.Item(1) 'gets the first document
```

Validation

One common task on documents is to validate them against an assigned schema or DTD. If the XML file has no schema or DTD already assigned, use `Document.AssignSchema` or `Document.AssignDTD` to add the necessary references to the document.

Examples:

```
objSpy.ActiveDocument.AssignSchema "C:\mySchema.xsd", False

  Or

objSpy.ActiveDocument.AssignDTD "C:\myDTD.dtd", False
```

If you want the user to select a schema or DTD, pass True as the second parameter to these functions to display a file-dialog. These methods only put the reference into the document and do not check the existence of the specified file. If the file path is not valid, the validation will fail.

After you have assigned a valid schema or DTD reference to your file, you are able to validate it with `Document.IsValid`. IsValid needs some out-parameters that must be declared as VARIANTs to be accessible from script languages like VBScript and JavaScript.

Example:

```
Dim bValid   As Boolean
Dim strMsg   As Variant
Dim nPos  As Variant
Dim objBadXMLData As Variant

bValid = objSpy.ActiveDocument.IsValid(strMsg, nPos, objBadXMLData)

If bValid = False Then
 a = MsgBox("The document is not valid:" & Chr(13) &
   strMsg & Chr(13) & "position: " & nPos &
   Chr(13) & "XMLData name: " & objBadXMLData.Name)
Else
 a = MsgBox("The document is valid")
End If
```

4.1.3 Error handling

The XMLSpy API returns errors in two different ways. Every API method returns an `HRESULT`. This return value informs the caller about any malfunctions during the execution of the method. If the call was successful, the return value is equal to `S_OK`. C/C++ programmers generally use `HRESULT` to detect any errors.

VisualBasic, scripting languages and other high-level development environments, don't give the programmer access to the returning `HRESULT` of a COM call. They use the second error raising mechanism also supported by the XMLSpy API, the `IErrorInfo` interface. If an error occurs the API creates a new object that implements the `IErrorInfo` interface. The development environment takes this interface, and fills its own error handling mechanism with the provided information.

The next paragraphs describes how to deal with errors raised from the XMLSpy API in different development environments.

VisualBasic

A common way to handle errors in VisualBasic is to define an error handler. This error handler can be set with the `On Error` statement. Usually the handler displays a error message and does some cleanup to avoid spare references and any kind of resource leaks. VisualBasic fills its own `Err` object with the information from the `IErrorInfo` interface.

Example:

```
Sub Validate()
 'place variable declarations here

 'set error handler
 On Error GoTo ErrorHandler

 'if IsValid() fails, program execution continues
            'at ErrorHandler:
 bValid = objSpy.ActiveDocument.IsValid(strMsg,nPos,objBadXMLData)

 'additional code comes here

 'exit
 Exit Sub

 ErrorHandler:
 Set objBadXMLData = Nothing

 MsgBox("Error: " & (Err.Number - vbObjectError) & Chr(13) &
  "Description: " & Err.Description)
End Sub
```

JavaScript
The Microsoft implementation of JavaScript (JScript) provides a try-catch mechanism to deal with errors raised from COM calls. It is very similar to the VisualBasic approach, because you also declare an error object containing the necessary information.

Example:

```
Function Validate()
{
 // please insert variable declarations here

 try{
  bValid = objSpy.ActiveDocument.IsValid(sMsg,nPos,objBad);
 }
 catch(Error) {
  objBad = Null;
  sError = Error.description;
  nErrorCode = Error.number;
  Return False;
 }

 Return bValid;
}
```

C/C++
C/C++ gives you easy access to the `HRESULT` of the COM call and to the `IErrorInterface`.

```
HRESULT hr;

// Call IsValid() from the XMLSpy API
If(FAILED(hr = ipDoc->IsValid(&varMsg,&varPos,&varBadData)))      {
 IErrorInfo *ipErrorInfo = Null;

 If(SUCCEEDED(::GetErrorInfo(0,&ipErrorInfo)))      {
  BSTRbstrDescr;
  ipErrorInfo->GetDescription(&bstrDescr);
```

```
      // handle Error information
      wprintf(L"Error message:\t%s\n",bstrDescr);
      ::SysFreeString(bstrDescr);

      // release Error info
      ipErrorInfo->Release();
    }
  }
```

4.1.4 Events

The automation interface of XMLSPY provides a large set of events to allow for a tight integration of XMLSPY and its clients. The way events can be used by clients depends on their technology used to connect to XMLSPY and on the clients programming language.

XMLSPY scripting environment - macros, forms and event handlers written with the FormEditor

The XMLSPY scripting environment automatically invokes the event handler with the corresponding name. Use the FormEditor to add code for those events you want to handle. The FormEditor predefines the correct signatures for all event handlers when you create a new project. For more information see the scripting environment.

VBScript

All event handler functions start with 'On_'. To stay compatible to existing events, their name might differ slightly from the name used on the connection point interface. Specification of event parameters is optional so that old event handlers that did not expect any parameters still work.

JScript

All event handler functions start with 'On_'. To stay compatible to existing events, their name might differ slightly from the name used on the connection point interface. Specification of event parameters is optional so that old event handlers that did not expect any parameters still work.

XMLSPY IDE Plugin

XMLSPY Plugins get informed about outstanding events by a call to the method IXMLSpyPlugIn.OnEvent. Different events are identified by a unique number. Parameters are passed to the event handler packed into a save-array.
Alternatively, Plugins can use the connection point mechanism to request events on specific objects like any other external client. See below for more information.

External clients

COM specifies the connection point mechanism to define a way how a client can register itself at a server for callbacks. 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.

VBA - VisualBasic for Applications

Use the Dim WithEvents statement to receive events from a dedicated object instance.

```
' tell VBA to connect to events once this objects gets created
Dim WithEvents objView As XMLSpyLib.AuthenticView
' initialize the object somewhere in your code
...
Set objView = objSpy.ActiveDocument.AuthenticView
...
' sample event handler for the OnMouseEvent of the object in objView
Private Function objView_OnMouseEvent (ByVal i_nXPos As Long, ByVal
i_nYPos As Long,
                                       ByVal i_eMouseEvent As
XMLSpyLib.SPYMouseEvent,
                                       ByVal i_pRange As
XMLSpyLib.IAuthenticRange ) As Boolean
    If (i_eMouseEvent = (XMLSpyLib.spyLeftButtonDownMask Or
XMLSpyLib.spyCtrlKeyDownMask)) Then
        On Error Resume Next
        i_pRange.Select
        objView_OnMouseEvent = True
    Else
        objView_OnMouseEvent = False
    End If
End Function
```

Windows Scripting Technologies - VBScript

Use the method WScript.ConnectObject 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")

' 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
```

Windows Scripting Technologies - JScript

Use the method WScript.ConnectObject to receive events.

```
// the event handler function
function DocEvent_OnBeforeCloseDocument(objDocument)
{
   WScript.Echo("received event - before closing document");
}

// create or connect to XMLSPY
try
{
   // create the environment and XMLSPY
   objWshShell = WScript.CreateObject("WScript.Shell");
   objFSO = WScript.CreateObject("Scripting.FileSystemObject");
   objSpy = WScript.GetObject("", "XMLSpy.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\\XMLSPY2004\\Examples\\OrgChart.xml", false);
WScript.ConnectObject(objDoc, "DocEvent_");

// keep running while waiting on the event
// in the meantime close this document in XMLSPY manually
WScript.Echo ("sleeping for 10 seconds ...");
WScript.Sleep (10000);

objDoc = null;
WScript.Echo ("stopped listening for event");
objSpy.Quit();
```

C++

The following is an excerpt of a C++ client using connection points via ATL

```
#import "XMLSpy.tlb"
using namespace XMLSpyLib;

static const int XMLSPYSE_APPLICATION_SOURCE_ID = 0;
class XMLSpySEApplicationEventPump;

// nested class to receive application events
typedef IDispEventImpl< XMLSPYSE_APPLICATION_SOURCE_ID,
                      XMLSpySEApplicationEventPump,
                      &DIID__IApplicationEvents,
                      &LIBID_XMLSpyLib, TYPELIB_MAJ, TYPELIB_MIN >
         XMLSpySEApplicationEventImpl;

// the class receiving the application events using a sink map
class XMLSpySEApplicationEventPump :
   public XMLSpySEApplicationEventImpl
{
   private:
         static _ATL_FUNC_INFO Info_OnBeforeOpenProject;

   public:
         XMLSpySEApplicationEventPump ();
         ~XMLSpySEApplicationEventPump();

   // connection point events on XMLSpySEApplicationEventImpl
   public:
         void __stdcall OnBeforeOpenProject (IFileSelectionDlg
*io_ipDlg);

   BEGIN_SINK_MAP(XMLSpySEApplicationEventPump)
         // OnBeforeOpenProject
         SINK_ENTRY_INFO(XMLSPYSE_APPLICATION_SOURCE_ID,
DIID__IApplicationEvents,
                              1, OnBeforeOpenProject,
&Info_OnBeforeOpenProject)
   END_SINK_MAP()
};

// we use here explicit declaration of event signature to avoid loading
the typelib
_ATL_FUNC_INFO XMLSpySEApplicationEventPump::Info_OnBeforeOpenProject =
   {CC_STDCALL, VT_EMPTY, 1, {VT_DISPATCH}};

XMLSpySEApplicationEventPump::XMLSpySEApplicationEventPump ()
{
   ...
   // get an IUnknown pointer to XMLSpy application from somewhere and
store it in ipXMLSPY
   HRESULT hRes = XMLSpySEApplicationEventImpl::DispEventAdvise(ipXMLSPY,
&DIID__IApplicationEvents);
   ...
}

XMLSpySEApplicationEventPump::~XMLSpySEApplicationEventPump()
{
   ...
   // disconnect from connection point
   HRESULT hRes =
XMLSpySEApplicationEventImpl::DispEventUnadvise(ipXMLSPY,
&DIID__IApplicationEvents);
   ...
}

// the handler for the OnBeforeOpenProject event
void __stdcall XMLSpySEApplicationEventPump::OnBeforeOpenProject
(IFileSelectionDlg *io_ipDlg)
{
   // do something with io_ipDlg here
}
```

4.1.5 Import and export of data

Before you implement your import and export tasks with the XMLSpy API, it is a good practice to test the connections, parameters, SQL queries and so on in XMLSpy. This way, you are able to verify the results and to make quick adjustments to all 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 (e.g. automatic key generation or number of sub-levels) to use from the XML-structure for the behaviour.

The properties in the `DatabaseConnection`, `TextImportExportSettings` and `ExportSettings` interfaces have default values. See the corresponding descriptions in the `Interfaces` chapter for further information.

Import from database
These 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:

   ```
   Dim objImpSettings As DatabaseConnection
   Set 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 = "C:\myDatabase.mdb"`

To complete the settings you create a **SQL select statement** to define the data to be queried:

- `objImpSettings.SQLSelect = "SELECT * FROM myTable"`

2. Call `Application.GetDatabaseImportElementList` to get a collection of the resulting columns of the SQL query:

```
Dim objElementList As ElementList
Set objElementList =
objSpy.GetDatabaseImportElementList(objImpSettings)
```

This collection gives you the opportunity to control which columns should be imported and what type the new elements will become. 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 type of the created 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 (Nothing in VisualBasic) as the second parameter to `ImportFromDatabase()` to import all columns as plain XML elements.

3. Start the import with `Application.ImportFromDatabase`:

```
Dim objImpDoc As Document
Set objImpDoc =
objSpy.ImportFromDatabase(objImpSettings,objElementList)
```

Import from Text

Importing data from a text file is similar to the import from a database. You must use other interfaces (described in steps 1-3 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.

```
Dim objImpSettings As TextImportExportSettings
Set 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 = "C:\myFile.txt"
objImpSettings.HeaderRow = False
```

Call `Application.GetTextImportElementList` to get a collection of all columns inside the text file:

```
Dim objElementList As ElementList
Set objElementList = objSpy.GetTextImportElementList(objImpSettings)
```

3. Start the import with `Application.ImportFromText`:

```
Dim objImpDoc As Document
Set objImpDoc = objSpy.ImportFromText(objImpSettings,objElementList)
```

Export to database

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 a ExportSettings object:

   ```
   Dim objExpSettings As ExportSettings
   Set objExpSettings = objSpy.GetExportSettings

   objExpSettings.CreateKeys = False
   objExpSettings.ExportAllElements = False
   objExpSettings.SubLevelLimit = 2
   ```

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.

Export to text

1. Use a `TextImportExportSettings` object and set the necessary properties.

2. Fill an `ExportSettings` object with the required values.
   See item number 2 from "Export to database" earlier on this page.

3. Build an element list with `Document.GetExportElementList`.
   See item number 3 from "Export to database" earlier on this page.

4. Call `Document.ExportToText` to execute the export.
   The description of the `ExportToText` method contains also a code example for a database export.

4.1.6 Using XMLData to modify document structure

XMLData gives you access to the elements of an currently open XML file. It enables you to perform all necessary modifications to the elements of the XML structure. The main functionality of `XMLData` is:

1. Access to the names and values of all kinds of elements (e.g. elements, attributes)

2. Creation of new elements of all kinds.

3. Insertion and appending of new elements.
4. Erasing of existing child elements.

Structure of XMLData

Before you can use the `XMLData` interface, you have to know how an existing XML file is mapped into a `XMLData` structure. One major thing you must be aware of is, that `XMLData` has no separate branch of objects for attributes.

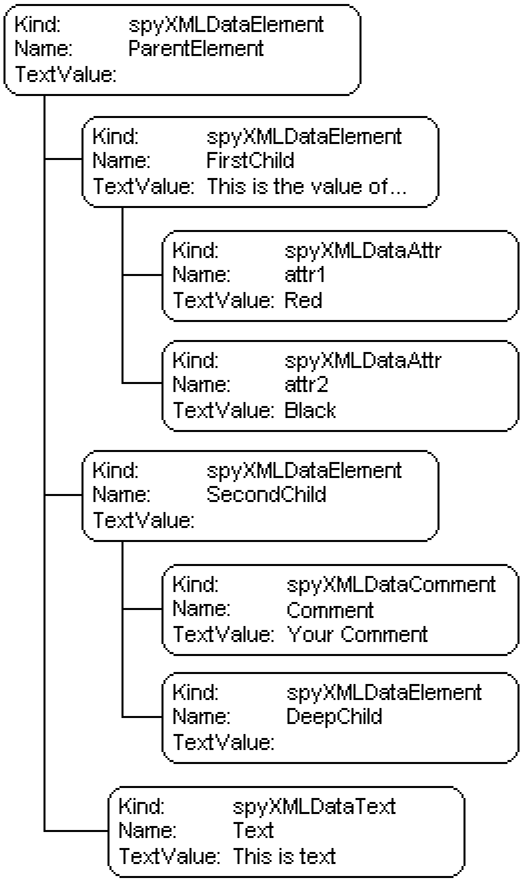
The attributes of an element are also children of the element. The `XMLData.Kind` property, gives you the opportunity to distinguish between the different types of children of an element.

Example:

This XML code,

```
<ParentElement>
 <FirstChild attr1="Red" attr2="Black">
  This is the value of FirstChild
 </FirstChild>
 <SecondChild>
  <!--Your Comment-->
  </DeepChild>
 </SecondChild>
 This is Text
</ParentElement>
```

is mapped to the following XMLData object structure:



The parent of all XML elements inside of a document is the property `Document.RootElement` . The first element of a document's content - without the XML prolog - is accessible with the property `Document.DataRoot`. Use one of these `XMLData` objects to get references to all other XML elements in the structure.

Name and value of elements

To get and to modify the name and value of all types of XML elements use the `XMLData.Name` and `XMLData.TextValue` properties. It is possible that several kinds of `XMLData` objects and empty elements do not have a text value associated.

Creation and insertion of new XMLData objects

The creation of a new XML language entity requires the following steps:

1. Create the new `XMLData` object:
   Use the `Document.CreateChild` method to create a new `XMLData` object. Set name and value after you have inserted the new XML entity (see point 3).
2. Find the correct location for the new XMLData object:
   To insert a new `XMLData` object you have to get a reference to the parent first. If the new child is to become the last child of the parent, use the `XMLData.AppendChild` method to insert the `XMLData` object. If the new child should be located elsewhere in the sequence of child objects, use the `XMLData.GetFirstChild` and `XMLData.GetNextChild` to move the iterator to the child before which the new child should be inserted.
3. Insert the new child with `XMLData.InsertChild`. The new child will be inserted immediately before the current child.

The following example adds a third child between <FirstChild> and the <SecondChild> element:

```
 Dim objParent As XMLData
 Dim objChild As XMLData
 Dim objNewChild As XMLData

 Set objNewChild = objSpy.ActiveDocument.CreateChild(spyXMLDataElement)

 'objParent is set to <ParentElement>
 'GetFirstChild(-1) gets all children of the parent element
 'and move to <SecondChild>
 Set objChild = objParent.GetFirstChild(-1)
 Set objChild = objParent.GetNextChild

 objParent.InsertChild objNewChild
 objNewChild.Name = "OneAndAHalf"
 Set objNewChild = Nothing
```

Child elements should be inserted in a special order. Please avoid inserting attributes into a sequence after any other child elements, i.e. make sure that attributes are placed first.

Copying of existing XMLData objects

If you want to insert existing XMLData objects at a different place in the same document or into another document you can't use the XMLData.InsertChild and XMLData.AppendChild methods. These methods only work for new XMLData objects.

Instead of using InsertChild or AppendChild you have to copy the object hierarchy manually. The following function written in JavaScript is an example for recursively copying XMLData:

```
// ----------------------------------------------------------
// XMLSpy scripting environment - GlobalDeclarations - JScript
// return a deep copy of the XMLData Object.
// the new object is owned by the document sepcified in objDoc
// ----------------------------------------------------------
function XMLDataDeepCopy(objXMLData, objDoc)
{
 // create new element of same kind
 var objNew = objDoc.CreateChild(objXMLData.Kind);

 // copy name, some element have default names and do not
 // allow to modifying these names.
 try
 { objNew.Name = objXMLData.Name; }
 catch (e)
 {
  if ((e.number & 0xffff) != 1513)      // modification operation not allowed
   throw(e);
 }

 // copy the text value
 objNew.TextValue = objXMLData.TextValue;

 // copy and insert all children
 if(objXMLData.HasChildren)
 {
  var objChild = objXMLData.GetFirstChild(-1);

  while(objChild)
  {
   try
```

```
        {
         objNew.AppendChild(XMLDataDeepCopy(objChild, objDoc));
         objChild = objXMLData.GetNextChild();
        }
        catch(e)
        {
         if ((e.number & 0xffff) == 1503)    // end of list
          objChild = null;
         else
          throw(e);
        }
      }
    }

   return objNew;
  }
```

Erasing of XMLData objects
XMLData provides two methods for the deletion of child objects, XMLData.EraseAllChildren and XMLData.EraseCurrentChild.

To erase XMLData objects you need access to the parent of the elements you want to remove. Use XMLData.GetFirstChild and XMLData.GetNextChild to get a reference to the parent XMLData object.

See the method descriptions of EraseAllChildren and EraseCurrentChild for examples how to erase XML elements.

4.1.7 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;
}
```

4.1.8 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**
`IsRowInsertEnabled`	**RowInsert**, superseded by `AuthenticRange.InsertRow`	Insert row operation
`IsRowAppendEnabled`	**RowAppend**, superseded by `AuthenticRange.AppendRow`	Append row operation
`IsRowDeleteEnabled`	**RowDelete**, superseded by `AuthenticRange.DeleteRow`	Delete row operation
`IsRowMoveUpEnabled`	**RowMoveUp**, superseded by `AuthenticRange.MoveRowUp`	Move XML data up one row
`IsRowMoveDownEnabled`	**RowMoveDown**, superseded by `AuthenticRange.MoveRowDown`	Move XML data down one row
`IsRowDuplicateEnabled`	**RowDuplicate**, superseded by `AuthenticRange.DuplicateRow`	Duplicate currently selected row

4.1.9 Obsolete Authentic View Editing operations

When XML data is displayed as data in Authentic View, it is possible to manipulate individual elements using standard editing operations such as cut, copy, and paste. However, not all XML data nodes can be edited. So, in order to carry out an editing operation, first a property is used to test whether editing is possible, and then a method is called to perform the editing operation.

The only method that does not have a test is the method `EditSelectAll`, which automatically selects all elements displayed in the document.

The following is a list of properties and methods that perform editing operations. Each property returns a BOOL, and the methods have no parameter.

Property	**Method**	**Editing operation**
`IsEditUndoEnabled`	**EditUndo**, superseded by `AuthenticView.Undo`	Undo an editing operation
`IsEditRedoEnabled`	**EditRedo**, superseded by `AuthenticView.Redo`	Redo an editing operation
`IsEditCopyEnabled`	**EditCopy**, superseded by `AuthenticRange.Copy`	Copy selected text to the clipboard

`IsEditCutEnabled`	**EditCut**, superseded by `AuthenticRange.Cut`	Cut selected text to the clipboard
`IsEditPasteEnabled`	**EditPaste**, superseded by `AuthenticRange.Paste`	Paste from clipboard to current cursor position
`IsEditClearEnabled`	**EditClear**, superseded by `AuthenticRange.Delete`	Clear selected text from XML document

4.2 Usage Examples

The automation interface of XMLSpy can be used in many different ways and accessed from different programming languages. In this section you will find sample code that show the capabilities of the new AuthenticView and AuthenticRange interfaces. These samples should make it easier for you to get started with your own project. Although they are more complex then those you will find in the descriptions of the various methods and properties, they are fragments and might require additional code or small adaptations in order for them to be properly executed in your environment.

The listed code samples are:

- JScript: Bubble Sort Dynamic Tables
- VBScript: Using object-level events

This section also briefly describes some operations that can be performed within Authentic View . This is to give you an idea of what is possible with the XMLSpy API.

4.2.1 JScript: 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 you 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 = objSpy = WScript.GetObject("", "XMLSpy.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");
}
```

4.2.2 VBScript: Using object-level events

Authentic view now 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 VB code snippet that shows how to connect to object-level events from within a VB project.

```
  '
```

```
----------------------------------------------------------------------------
' VB code snippet - connecting to object level events
'
----------------------------------------------------------------------------
Dim objSpy As XMLSpyLib.Application
' use VBA keyword WithEvents to automatically connect to object-level event
Dim WithEvents objView As XMLSpyLib.AuthenticView

' this is the event callback routine that will be 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

' this is the event callback routine that will be 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

' this is the event callback routine that will be connected to the
' OnMouseEvent event of object objView.
' If you click with the left mouse button while pressing a control key,
' the current selection will be set to the tag below the current
' mouse cursor position
Private Function objView_OnMouseEvent(ByVal i_nXPos As Long, ByVal i_nYPos
As Long, ByVal i_eMouseEvent As XMLSpyLib.SPYMouseEvent, ByVal i_pRange As
XMLSpyLib.IAuthenticRange) As Boolean
    If (i_eMouseEvent = (XMLSpyLib.spyLeftButtonDownMask Or
XMLSpyLib.spyCtrlKeyDownMask)) Then
        On Error Resume Next
        i_pRange.Select
        objView_OnMouseEvent = True
    Else
        objView_OnMouseEvent = False
    End If
End Function

' connect to running XMLSpy application
' this code will most likely be in some Form_Load() subroutine
Set objSpy = GetObject("", "XMLSpy.Application")
objSpy.ShowApplication (True)
If (objSpy.ActiveDocument Is Nothing) Then
    Dim objDoc As XMLSpyLib.Document
    Rem replace path below With a valid absolute path On your machine
    Set objDoc = objSpy.Documents.OpenFile("InputData\orgchart.xml", False)
End If

If (objSpy.ActiveDocument.CurrentViewMode <> spyViewContent) Then
    objSpy.ActiveDocument.SwitchViewMode (spyViewContent)
End If
Set objView = objSpy.ActiveDocument.AuthenticView
' now the object objView is set and selection change events are received

' continue here with something useful ...
' and serve the windows message loop

' if you want to stop receiving events from object objView
' add the following line somewhere appropriate
Set objView = Nothing
```

4.3 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)
      - XMLData
  - Dialogs
    - CodeGeneratorDlg
    - FileSelectionDlg
    - SchemaDocumentationDlg
  - 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.

4.3.1 Application

See also

Methods
GetDatabaseImportElementList
GetDatabaseSettings
GetDatabaseTables
ImportFromDatabase

GetTextImportElementList
GetTextImportExportSettings
ImportFromText

ImportFromWord

ImportFromSchema

GetExportSettings

NewProject
OpenProject

AddMacroMenuItem
ClearMacroMenu

ShowForm

ShowApplication

URLDelete
URLMakeDirectory

Quit

Properties
Application
Parent

ActiveDocument
Documents

CurrentProject

Dialogs

WarningNumber
WarningText

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.DialogAction` property of *io_objDialog* at its default value `spyDialogOK`. To abort the opening of the document set this property to `spyDialogCancel`.

Examples
Given below are examples of how this event can be scripted.

XMLSpy scripting environment - VBScript:

```
Function On_BeforeOpenDocument(objDialog)
End Function
```

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.DialogAction property of *io_objDialog* at its default value spyDialogOK. To abort the opening of the project set this property to spyDialogCancel.

Examples
Given below are examples of how this event can be scripted.

XMLSpy scripting environment - VBScript:

```
Function On_BeforeOpenProject(objDialog)
End Function
```

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)
End Function
```

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)
End Function
```

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)

Return Value

Description
Adds an 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 with 16 items.

Application

See also

Property: `Application` as `Application` (read-only)

Description
Access 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.

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.

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.

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.

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.

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 ,pElementList 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.

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.

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 null 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.

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.

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
Access 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.

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.

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 `strFormName`.

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.

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 `strURL`.

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 `Boolean`

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.

4.3.2 AuthenticDataTransfer

Renamed from DocEditDataTransfer to AuthenticDataTransfer

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

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 IUnkown 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.

4.3.3 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		Methods
Application	Clone	MoveBegin
FirstTextPosition	CollapsToBegin	MoveEnd
FirstXMLData	CollapsToEnd	NextCursorPosition
FirstXMLDataOffset	ExpandTo	PreviousCursorPosition
LastTextPosition	Goto	Select
LastXMLData	GotoNext	SelectNext
LastXMLDataOffset	GotoPrevious	SelectPrevious
Parent	IsEmpty	SetFromRange
	IsEqual	

The following table lists the content modification methods, most of which can be found on the right/button mouse menu.

Properties	Edit operations	Dynamic table operations
Text	Copy	AppendRow
	Cut	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	Attributes	Entities
CanPerformActionWith	GetElementAttributeValue	GetEntityNames
CanPerformAction	GetElementAttributeNames	InsertEntity
PerformAction	GetElementHierarchy	
	HasElementAttribute	
	IsTextStateApplied	
	SetElementAttributeValue	

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
Access 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: PerformActionWith (*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
```

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 test position by 1, has the same effect as the cursor-right key. Decrementing the test position by 1 has the same effect as the cursor-left key.

If you set FirstTextPosition to a value greater than the current LastTextPosition, 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.

Examples
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 `InsertEntity`.

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)
     objRange.Select()
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.

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 then the current `FirstXMLData` / `FirstXMLDataOffset`, the latter gets moved to the new end position.

HINT: You can use the `FirstXMLData` and `LastXMLData` properties to directly access and manipulate the underlying XML document in those cases, where the methods available with the `AuthenticRange` object are not sufficient.

Errors

2001	The authentic range object, or its related view object is not valid.
2005	Invalid address for the return parameter was specified.
2008	Internal error
2009	The `XMLData` object cannot be accessed.

LastXMLDataOffset

See also

Property: `LastXMLDataOffset` as Long

Description
Set or get the cursor position inside `LastXMLData` element for the end of the range.

Offset positions are based on the characters returned by the `Text` property and start with 0. When setting a new offset, use -1 to set the offset to the last possible position in the element. The following cases require specific attention:

- The textual form of entries in Combo Boxes, Check Boxes and similar controls can be different from what you see on the screen. Although, the data offset is based on this text, there only two valid offset positions, one at the beginning and one at the end of the entry. An attempt to set the offset to somewhere in the middle of the entry, will result in the offset being set to the end.

- The textual form of XML Entities might differ in length from their representation on the screen. The offset is based on this textual form.

If LastXMLData / LastXMLDataOffset selects a position before FirstXMLData / FirstXMLDataOffset, the latter gets moved to the new end position.

Errors

2001 The authentic range object, or its related view object is not valid.
2005 Invalid offset was specified.
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 "Ooops"
End If
```

MoveBegin

See also

Method: MoveBegin (*eKind* as SPYAuthenticElementKind, *nCount* as Long) as AuthenticRange

Description

Move the beginning of the range to the beginning of the nCount element of type eKind. Counting starts at the current beginning of the range object.

Use positive numbers for nCount to move towards the document end, use negative numbers to move towards document beginning. The end of the range stays unmoved, unless the new beginning would be larger than it. In this case, the end is moved to the new beginning. The method returns the modified range object.

Errors

2001 The authentic range object, or its related view object is no longer valid.

2003	Target lies after end of document.
2004	Target lies before beginning of document.
2005	Invalid element kind specified. Invalid address for the return parameter was specified.

MoveEnd

See also

Method: `MoveEnd` (*`eKind`* as `SPYAuthenticElementKind`, *`nCount`* as Long) as `AuthenticRange`

Description

Move the end of the range to the begin of the `nCount` element of type `eKind`. Counting starts at the current end of the range object.

Use positive numbers for `nCount` to move towards the document end, use negative numbers to move towards document beginning. The beginning of the range stays unmoved, unless the new end would be less than it. In this case, the beginning gets moved to the new end. The method returns the modified range object.

Errors

2001	The authentic range object, or its related view object is no longer valid.
2003	Target lies after end of document.
2004	Target lies before begin of document.
2005	Invalid element kind specified. Invalid address for the return parameter was specified.

MoveRowDown

See also

Method: `MoveRowDown()` as Boolean

Description

If the beginning of the range is inside a dynamic table and selects a row which is not the last row in this table, this method swaps this row with the row immediately below. The selection of the range moves with the row, but does not otherwise change. The function returns *true* if the move operation was successful, otherwise *false*.

Errors

2001	The authentic range object or its related view object is no longer valid.
2005	Invalid address for the return parameter was specified.

MoveRowUp

See also

Method: `MoveRowUp()` as Boolean

Description

If the beginning of the range is inside a dynamic table and selects a row which is not the first row in this table, this method swaps this row with the row above. The selection of the range moves with the row, but does not change otherwise. The function returns *true* if the move operation was successful, otherwise *false*.

Errors

2001 The authentic range object, or its related view object is no longer valid.
2005 Invalid address for the return parameter was specified.

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
                  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.

Text

See also

Property: `Text` as String

Description
Set or get the textual content selected by the range object.

The number of characters retrieved are not necessarily identical, as there are text cursor positions between the beginning and end of the selected range. Most document elements support an end cursor position different to the beginning cursor position of the following element. Drop-down lists maintain only one cursor position, but can select strings of any length. In the case of radio buttons and check boxes, the text property value holds the string of the corresponding XML element.

If the range selects more then one element, the text is the concatenation of the single texts. XML entities are expanded so that '&' is expected as '&amp;'.

Setting the text to the empty string, does not delete any XML elements. Use `Cut`, `Delete` or `PerformAction` instead.

Errors

2001	The authentic range object or its related view object is no longer valid.
2005	Invalid address for a return parameter was specified.

4.3.4 AuthenticView

See also

Properties	**Methods**	**Events**
`Application`	`Goto`	`OnBeforeCopy`

AsXMLString
DocumentBegin
DocumentEnd
Event
MarkupVisibility
Parent
Selection
XMLDataRoot
WholeDocument

IsRedoEnabled
IsUndoEnabled
Print
Redo
Undo
UpdateXMLInstanceEntities

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 <> spyViewContent) Then
            If (Not objDocument.SwitchViewMode (spyViewContent)) 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
```

Events

OnBeforeCopy

See also

Event: `OnBeforeCopy()` as Boolean

XMLSpy scripting environment - VBScript:

```
Function On_DocEditCopy()
     ' On_BeforeCopy = False ' to disable operation
End Function
```

XMLSpy scripting environment - JScript:

```
function On_DocEditCopy()
{
     // return false; /* to disable operation */
}
```

XMLSpy IDE Plugin:

```
IXMLSpyPlugIn.OnEvent (21, ...)  // nEventId = 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_DocEditCut()
     ' On_BeforeCopy = False ' to disable operation
End Function
```

XMLSpy scripting environment - JScript:

```
function On_DocEditCut()
{
     // 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_DocEditClear()
     ' On_BeforeCopy = False ' to disable operation
End Function
```

XMLSpy scripting environment - JScript:

```
function On_DocEditClear()
{
     //returnfalse; /*todisableoperation*/
}
```

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_DocEditDrop(nXPos, nYPos, objRange, objData)
     ' On_BeforeCopy = False ' to disable operation
End Function
```

XMLSpy scripting environment - JScript:

```
function On_DocEditDrop(nXPos, nYPos, objRange, objData)
{
     //returnfalse; /*todisableoperation*/
}
```

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 cancle 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_DocEditPaste(objData, strType)
     ' On_BeforeCopy = False ' to disable operation
End Function
```

XMLSpy scripting environment - JScript:

```
function On_DocEditPaste(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.

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_DocEditDragOver(nXPos, nYPos, eMouseEvent, objRange, objData)
     ' On_BeforeCopy = False ' to disable operation
End Function
```

XMLSpy scripting environment - JScript:

```
function On_DocEditDragOver(nXPos, nYPos, eMouseEvent, objRange, objData)
{
     // return 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_DocEditKeyboardEvent(eKeyEvent, nKeyCode, nVirtualKeyStatus
)
     ' On_DocEditKeyboardEvent = True ' to cancel bubbling of event
End Function
```

XMLSpy scripting environment - JScript:

```
function On_DocEditKeyboardEvent(eKeyEvent, nKeyCode, nVirtualKeyStatus)
{
     // return false; /* 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.

REMARK: The following events from the scripting environment and IDE Plugin of XMLSpy are still supported but become obsolete with this event:

`On_DocEditKeyUp()`	`IXMLSpyPlugIn.OnEvent` (13, ...)	// nEventId= 13
`On_DocEditKeyDown()`	`IXMLSpyPlugIn.OnEvent` (12, ...)	// nEventId= 12
`On_DocEditKeyPressed()`	`IXMLSpyPlugIn.OnEvent` (14, ...)	// nEventId= 14

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 OnKeyboard
' event of object objView
Private Function objView_OnKeyboardEvent(ByVal i_keyEvent As Long, ByVal
io_pnKeyCode As Long, ByVal i_nVirtualKeyStatus As Long) As Boolean
    If ((i_keyEvent = XMLSpyLib.spyKeyUp) And ((i_nVirtualKeyStatus And
XMLSpyLib.spyCtrlKeyMask) <> 0)) Then
        MsgBox ("Ctrl " & io_pnKeyCode & " pressed")
        objView_OnKeyboardEvent = True
    Else
        objView_OnKeyboardEvent = False
    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
```

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_DocEditMouseEvent(nXPos, nYPos, eMouseEvent, objRange)
     ' On_DocEditMouseEvent = True ' to cancel bubbling of event
End Function
```

XMLSpy scripting environment - JScript:

```
function On_DocEditMouseEvent(nXPos, nYPos, eMouseEvent, objRange)
{
     // return false; /* 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*.

REMARK: The following events from the scripting environment and IDE Plugin of XMLSpy are still supported but become obsolete with this event:

```
On_DocEditMouseMove()              IXMLSpyPlugIn.OnEvent (15, ...)    //
```

nEventId=15

On_DocEditButtonUp() IXMLSpyPlugIn.OnEvent (16, ...) //
nEventId=16

On_DocEditButtonDown() IXMLSpyPlugIn.OnEvent (17, ...) //
nEventId=17

On_DocEditButtonDoubleClick() IXMLSpyPlugIn.OnEvent (24, ...) //
nEventId=24

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 OnMouseEvent
' event of object objView. If you click with the left mouse button
' while pressing a control key, the current selection will be set
' to the tag below the current mouse cursor position
Private Function objView_OnMouseEvent(ByVal i_nXPos As Long, ByVal i_nYPos
As Long, ByVal i_eMouseEvent As XMLSpyLib.SPYMouseEvent, ByVal i_pRange As
XMLSpyLib.IAuthenticRange) As Boolean
    If (i_eMouseEvent = (XMLSpyLib.spyLeftButtonDownMask Or
XMLSpyLib.spyCtrlKeyDownMask)) Then
        On Error Resume Next
        i_pRange.Select
        objView_OnMouseEvent = True
    Else
        objView_OnMouseEvent = False
    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
```

OnSelectionChanged

See also

Event: OnSelectionChanged (*objNewSelection* as AuthenticRange)

XMLSpy scripting environment - VBScript:

Function On_DocEditSelectionChanged (*objNewSelection*)
End Function

XMLSpy scripting environment - JScript:

function On_DocEditSelectionChanged (*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
```

Application

See also

Property: `Application` as `Application` (read-only)

Description

Access 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.

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.

Event

See also

Property: Event as AuthenticEvent (read-only)

Description
This property gives access to parameters of the last event in the same way as OldAuthenticView.event does. Since all events for the scripting environment and external clients are now available with parameters this Event property should only be used from within IDE-Plugins.

Errors

2000	The authentic view object is no longer valid.
2005	Invalid address for the return parameter was specified.

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
```

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.

4.3.5 GenerateSampleXMLDlg

See also

Properties and Methods

Standard automation properties
Application
Parent

NonMandatoryAttributes
NonMandatoryElements
TakeFirstChoice
RepeatCount
FillWithSampleData

Description
Used to set the parameters for the generation of sample XML instances based on a W3C schema or DTD.

Application

See also

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

See also

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

See also

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

See also

Property: `NonMandatoryElements` as Boolean

Description
If true elements 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.

TakeFirstChoice

See also

Property: `TakeFirstChoice` as Boolean

Description
If true the first entry is taken for a mandatory choice element.

Errors

2200	The object is no longer valid.
2201	Invalid address for the return parameter was specified.

RepeatCount

See also

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

See also

Property: `FillWithSampleData` as Boolean

Description
Set the property to true to generate sample data for elements and attributes.

Errors

2200	The object is no longer valid.
2201	Invalid address for the return parameter was specified.

4.3.6 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
TemplateFileName
CompatibilityMode

Settings for C++ code
CPPSettings_DOMType
CPPSettings_LibraryType
CPPSettings_UseMFC
CPPSettings_GenerateVC6ProjectFile
CPPSettings_GenerateVSProjectFile

Settings for C# code
CSharpSettings_ProjectType

Dialog handling for above code generation properties
PropertySheetDialogAction

Output path selection properties
OutputPath
OutputPathDialogAction

Presentation of result
OutputResultDialogAction

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_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 `spyVisualStudio2003Project` (=1) and `spyVisualStudio2005Project` (=4) 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.

4.3.7 DatabaseConnection

See also

Properties for import and export
File or
ADOConnection or
ODBCConnection

Properties for import only
DatabaseKind
SQLSelect
AsAttributes
ExcludeKeys
IncludeEmptyElements
NumberDateTimeFormat

Properties for export only
CreateMissingTables
CreateNew
TextFieldLen

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.

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.

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.

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.

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.

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 long

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.

4.3.8 Dialogs

See also

Properties and Methods

Standard automation properties
Application
Parent

Various dialog objects
CodeGeneratorDlg
FileSelectionDlg
SchemaDocumentationDlg
GenerateSampleXMLDlg

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

1111	The Dialogs object or one of its parents is no longer valid.
1100	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

1111	The Dialogs object or one of its parents is no longer valid.
1100	Invalid address for the return parameter was specified.

Parent

See also

Property: `Parent` as `Application` (read-only)

Description
Access the XMLSpy application object.

Errors

1111	The object is no longer valid.
1100	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

1111	The Dialogs object or one of its parents is no longer valid.
1100	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

1111	The Dialogs object or one of its parents is no longer valid.
1100	Invalid address for the return parameter was specified.

4.3.9 Document

See also

Properties and Methods

Standard automation properties
`Application`
`Parent`

Various document properties and methods
`SetActiveDocument`
`Encoding`
`SetEncoding (obsolete)`

XML validation
IsWellFormed
IsValid
SetExternalIsValid

Document conversion and transformation
AssignDTD
AssignSchema
AssignXSL
AssignXSLFO
ConvertDTDOrSchema
GenerateDTDOrSchema
CreateSchemaDiagram
ExecuteXQuery
TransformXSL
TransformXSLFO
GenerateProgramCode (Enterprise Edition only)
GenerateSchemaDocumentation
GenerateSampleXML

Document export
GetExportElementList
ExportToText
ExportToDatabase

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)
End Function

' 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)
End Function
```

XMLSpy scripting environment - JScript:

```
function 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 FileSelectionDlg.DialogAction 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 closing of document
End Function
```

XMLSpy scripting environment - JScript:

```
function On_BeforeCloseDocument(objDocument)
{
     // return false; /* to prohibit closing 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
End Function
```

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 SetExternalIsValid 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)
End Function
```

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)
End Function
```

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
Access 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.

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 <> spyViewContent) Then
```

```
             If (Not objDocument.SwitchViewMode (spyViewContent)) 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.

Errors

1400	The object is no longer valid.
1412	Error during conversion.

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.

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`.

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 XMLSpyDocument.RootElement to get the root element of the whole document including XML prolog data.

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 *spyViewContent*. 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.

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).

Errors

1400	The object is no longer valid.
1407	Invalid parameter or invalid address for the return parameter was specified.
1416	Error during export.

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 .

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.

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
```

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.

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`
Set to 0 (zero) for unlimited.

`nDetection`
Specifies attribute/element detection.

`nFrequentElements`
Create complex elements as elements or complex types.

Description
To generate a DTD or schema from the current document use the `GenerateDTDOrSchema` method.

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 MS-Word or HTML format. The parameter objDlg is used to parameterize the generation process. Use `Dialogs.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.

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

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.

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.

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` and `nErrorPos` give you the same information as XMLSpy if you validate the file within the editor.

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

2400	The document object is no longer valid.
2401	Invalid address for the return parameter was specified.

Path

See also

Property: `Path` as String (read-only)

Description
Use this property to retrieve the path - not including the file name - of the document file. To change the file name and path for a document use the property FullName.

Errors

1400	The document object is no longer valid.
1407	Invalid address for the return parameter was specified.

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.

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.

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

SetExternalIsValid

See also

Method: `SetExternalIsValid` (*`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.

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.

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.

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.

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 TextView if the current XML text is not well-formed.

Available with TypeLibrary version 1.5

Errors

1400	The document object is no longer valid.

4.3.10 Documents

See also

Properties

Count
Item

Methods

NewFile
OpenFile
OpenURL
OpenURLDialog
NewFileFromText

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.

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.

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.

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 , *nLoa*SPYURLTypes*ing* 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 authentification. 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 authentification. 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.

4.3.11 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 retuned 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.

4.3.12 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 Document.GetExportElementList.

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 th 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 Document.GetExportElementList.

4.3.13 ExportSettings

See also

Properties

ElementList

EntitiesToText

ExportAllElements
SubLevelLimit

FromAttributes
FromSingleSubElements
FromTextValues

CreateKeys
IndependentPrimaryKey

Namespace

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`.

FromAttributes

See also

Property: `FromAttributes` as Boolean

Description
Set `FromAttributes` to false if no export data should be created from attributes. Default is True.

FromSingleSubElements

See also

Property: `FromSingleSubElements` as Boolean

Description
Set `FromSingleSubElements` to false if no export data should be created from elements. Default is True.

FromTextValues

See also

Property: `FromTextValues` as Boolean

Description
Set `FromTextValues` 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`.

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`.

4.3.14 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.

4.3.15 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 ' to prohibit dragging
End Function
```

XMLSpy scripting environment - JScript:

```
function On_BeforeDrag()
{
      // return false; /* to prohibit dragging */
}
```

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 ' to prohibit dropping
End Function
```

XMLSpy scripting environment - JScript:

```
function On_BeforeDrop(objXMLData)
{
      // return false; /* to prohibit dropping */
}
```

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 ' to prohibit editing the field
  End Function
```

XMLSpy scripting environment - JScript:

```
  function On_BeforeStartEditing(objXMLData,bEditingName)
  {
        // return false; /* to prohibit editing the field */
  }
```

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, it 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)
  End Function
```

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)
End Function
```

XMLSpy scripting environment - JScript:

```
function On_FocusChanged(objXMLData,bSetFocus,bEditingName)
{
}
```

XMLSpy IDE Plugin:

```
IXMLSpyPlugIn.OnEvent (3, ...)    //nEventId=3
```

Description
This event gets fired whenever a grid cell receives or looses 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.

4.3.16 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`

`IncludeEverything`
`IncludeIndex`
`IncludeGlobalElements`
`IncludeLocalElements`
`IncludeGroups`
`IncludeComplexTypes`
`IncludeSimpleTypes`
`IncludeAttributeGroups`
`IncludeRedefines`

`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 OutputFile property to a valid file path. Use OptionsDialogAction, OutputFileDialogAction and ShowProgressBar to specify the level of user interaction desired. You can use IncludeEverything and AllDetails to set whole option groups at once or the individual properties to operate on a finer granularity.

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.

IncludeEverything

See also

Method: IncludeEverything (i_bInclude as Boolean)

Description
Use this method to mark or unmark all include options.

Errors

2900	The object is no longer valid.

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.

IncludeAttributeGroups

See also

Property: IncludeAttributeGroups as Boolean

Description

Set this property to true, to include attribute groups into the schema documentation. The property is initialized with the value used during the last call to Document.GenerateSchemaDocumentation.

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 into the schema documentation. The property is initialized with the value used during the last call to Document.GenerateSchemaDocumentation.

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 into the schema documentation. The property is initialized with the value used during the last call to Document.GenerateSchemaDocumentation.

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 into the schema documentation. The property is initialized with the value used during the last call to `Document.GenerateSchemaDocumentation`.

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 into the schema documentation. The property is initialized with the value used during the last call to `Document.GenerateSchemaDocumentation`.

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 into the schema documentation. The property is initialized with the value used during the last call to `Document.GenerateSchemaDocumentation`.

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 into the schema documentation. The property is initialized with the value used during the last call to `Document.GenerateSchemaDocumentation`.

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 into the schema documentation. The property is initialized with the value used during the last call to `Document.GenerateSchemaDocumentation`.

Errors

2900	The object is no longer valid.
2901	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 `Document.GenerateProgramCode`.

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 `SPYSchemaDocumentationFormat`

Description
Defines the kind of documentation that will be generated, either HTML style or MS-Word. The property gets initialized with the value used during the last call to `Document.GenerateSchemaDocumentation`.

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 `Document.GenerateSchemaDocumentation`.

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 `Document.GenerateSchemaDocumentation`.

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 `Document.GenerateSchemaDocumentation`.

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 `Document.GenerateSchemaDocumentation`.

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 `Document.GenerateSchemaDocumentation`.

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 `Document.GenerateSchemaDocumentation`.

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 `Document.GenerateSchemaDocumentation`.

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 `Document.GenerateSchemaDocumentation`.

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 `Document.GenerateSchemaDocumentation`.

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 `Document.GenerateSchemaDocumentation`.

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 `Document.GenerateSchemaDocumentation`.

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 `Document.GenerateSchemaDocumentation`.

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 `Document.GenerateSchemaDocumentation`.

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 `Document.GenerateSchemaDocumentation`.

Errors

2900	The object is no longer valid.
2901	Invalid address for the return parameter was specified.

4.3.17 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.

4.3.18 SpyProjectItem

See also

Methods
Open

Properties
ChildItems
ParentItem
FileExtensions
ItemType
Name
Path
ValidateWith
XMLForXSLTransformation
XSLForXMLTransformation
XSLTransformationFileExtension
XSLTransformationFolder

Description
SpyProjectItem 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.

4.3.19 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.

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 authentification. 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.

4.3.20 TextImportExportSettings

See also

Properties for import only
ImportFile

Properties for export only
DestinationFolder
FileExtension

Properties for import and export
HeaderRow
FieldDelimiter
EnclosingCharacter
Encoding
EncodingByteOrder

Description
`TextImportExportSettings` contains options common to text import and export functions.

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.

4.3.21 XMLData

See also

Properties
`Kind`
`Name`
`TextValue`

`HasChildren`
`MayHaveChildren`
`Parent`

Methods
`GetFirstChild`
`GetNextChild`
`GetCurrentChild`

`InsertChild`
`AppendChild`

`EraseAllChildren`
`EraseCurrentChild`

IsSameNode

CountChildren
CountChildrenKind

GetChild
GetChildKind

HasChildrenKind

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.

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 Document.RootElement at its top. To get the top element of the document content - ignoring the XML header - use Document.DataRoot. 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
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
```

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.

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`.

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);
}
```

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.

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.

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
1900	Document must not be modified

Examples
See Using XMLData to modify document structure .

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.

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 .

4.4 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.

4.4.1 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 beend 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.

4.4.2 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.

4.4.3 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.

4.5 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.

4.5.1 SPYAuthenticActions

Description
Actions that can be performed on AuthenticRange objects.

Possible values:

spyAuthenticInsertAt	= 0
spyAuthenticApply	= 1
spyAuthenticClearSurr	= 2
spyAuthenticAppend	= 3
spyAuthenticInsertBefore	= 4
spyAuthenticRemove	= 5

4.5.2 SPYAuthenticDocumentPosition

Description
Relative and absolute positions used for navigating with AuthenticRange objects.

Possible values:

spyAuthenticDocumentBegin	= 0
spyAuthenticDocumentEnd	= 1
spyAuthenticRangeBegin	= 2
spyAuthenticRangeEnd	= 3

4.5.3 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

4.5.4 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

4.5.5 SPYAuthenticMarkupVisibility

Description
Enumeration values to customize the visibility of markup with MarkupVisibility.

Possible values:

spyAuthenticMarkupHidden	= 0
spyAuthenticMarkupSmall	= 1
spyAuthenticMarkupLarge	= 2
spyAuthenticMarkupMixed	= 3

4.5.6 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

4.5.7 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`

4.5.8 SPYDOMType

Description
Enumeration values to parameterize generation of C++ code from schema definitions.

Possible values:

spyDOMType_msxml4	= 0
spyDOMType_xerces	= 1

4.5.9 SPYDTDSchemaFormat

Description
Enumeration to identify the different schema formats.

Possible values:

spyDTD	= 0
spyW3C	= 1

4.5.10 SPYEncodingByteOrder

Description
Enumeration values to specify encoding byte ordering for text import and export.

Possible values:

spyNONE	= 0
spyLITTLE_ENDIAN	= 1
spyBIG_ENDIAN	= 2

4.5.11 SPYExportNamespace

Description
Enumeration type to configure handling of namespace identifiers during export.

Possible values:

spyNoNamespace	= 0
spyReplaceColonWithUnderscore	= 1

4.5.12 SPYFrequentElements

Description
Enumeration value to parameterize schema generation.

Possible values:

spyGlobalElements	= 0
spyGlobalComplexType	= 1

4.5.13 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

4.5.14 SPYLibType

Description
Enumeration values to parameterize generation of C++ code from schema definitions.

Possible values:

spyLibType_static	= 0
spyLibType_dll	= 1

4.5.15 SPYLoading

Description

Enumeration values to define loading behaviour of URL files.

Possible values:

spyUseCacheProxy	= 0
spyReload	= 1

4.5.16 SPYMouseEvent

Description

Enumeration type that defines the mouse status during a mouse event. Use the enumeration values as bitmasks rather then 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

4.5.17 SPYNumberDateTimeFormat

Description

Enumeration value to configure database connections.

Possible values:

spySystemLocale	= 0

spySchemaCompatible	= 1

4.5.18 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

4.5.19 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

4.5.20 SPYProjectType

Description
Enumeration values to parameterize generation of C# code from schema definitions.

Possible values:

spyVisualStudioProject	= 0
spyVisualStudio2003Project	= 1
spyBorlandProject	= 2
spyMonoMakefile	= 3
spyVisualStudio2005Project	= 4

4.5.21 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

4.5.22 SPYSchemaDocumentationFormat

Description
Enumeration values to parameterize generation of schema documentation. These values are used in `SchemaDocumentationDialog.OutputFormat`.

Possible values:

spySchemaDoc_HTML	= 0
spySchemaDoc_MSWord	= 1

4.5.23 SPYTextDelimiters

Description
Enumeration values to specify text delimiters for text export.

Possible values:

spyTabulator	= 0
spySemicolon	= 1
spyComma	= 2
spySpace	= 3

4.5.24 SPYTextEnclosing

Description
Enumeration value to specify text enclosing characters for text import and export.

Possible values:

spyNoEnclosing	= 0
spySingleQuote	= 1
spyDoubleQuote	= 2

4.5.25 SPYTypeDetection

Description
Enumeration to select how type detection works during GenerateDTDOrSchema.

Possible values:

spyBestPossible	= 0
spyNumbersOnly	= 1
spyNoDetection	= 2

4.5.26 SPYURLTypes

Description
Enumeration to specify different URL types.

Possible values:

spyURLTypeAuto	= -1
spyURLTypeXML	= 0
spyURLTypeDTD	= 1

4.5.27 SPYViewModes

Description
Enumeration values that define the different view modes for XML documents. The mode *spyViewContent(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	

4.5.28 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 then 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

4.5.29 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
spyXMLDataP	= 9
spyXMLDataDefDoctype	= 10
spyXMLDataDefExternalID	= 11
spyXMLDataDefElement	= 12
spyXMLDataDefAttlist	= 13
spyXMLDataDefEntity	= 14
spyXMLDataDefNotation	= 15
spyXMLDataKindsCount	= 16

5 Using the XMLSpy API with Java

The XMLSpy API has an interface built up of Java classes, each of which corresponds to an object in the XMLSpy 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 XMLSpy API objects themselves, see the XMLSpy 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`
      - `SpyXMLData`
  - `SpyDialogs`
    - `SpyCodeGeneratorDlg`
    - `SpyFileSelectionDlg`
    - `SpySchemaDocumentationDlg`
  - `SpyDatabaseConnection`
  - `SpyElementList`
  - `SpyElementListItem`
  - `SpyExportSettings`
  - `SpyTextImportExportSettings`

Implementation of COM properties in Java

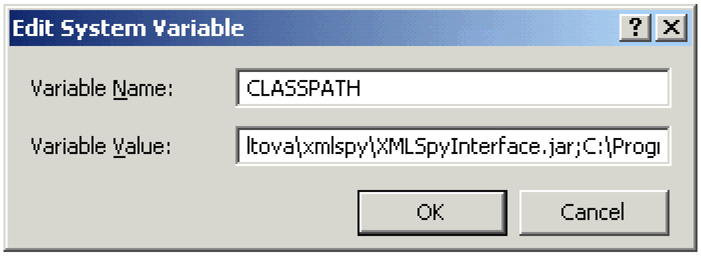
Properties in Java have been defined to include both a **`set`** and **`get`** method (`set` if it is allowed by the COM implementation). For example, the COM class `Document` contains the **`GridView`** property. In Java the method is called `SpyDoc` and the property is defined as a **`GetGridView`** 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 2000 and 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"`.


Edit System Variable

Variable Name: CLASSPATH

Variable Value: ltova\xmlspy\XMLSpyInterface.jar;C:\Progı

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).

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 = new SpyApplication();
  app.ShowApplication( true );

  SpyDoc doc = app.GetDocuments().OpenFile("C:\\Program Files\\Altova\\xmlspy
2006\\Examples\\OrgChart.xml", true );

  doc.SwitchViewMode(SPYViewModes.spyViewGrid);
  SpyXMLData oData = doc.GetRootElement();
  SpyXMLData oNewChild = doc.CreateChild(SPYXMLDataKind.spyXMLDataElement);

  oNewChild.SetName( "NewChild" );
  oNewChild.SetTextValue("newVaLuE");
  oData.AppendChild(oNewChild);
  oNewChild.ReleaseInstance();
  oData.ReleaseInstance();


  if ( doc.IsValid(  ) == false )
  {
   // is to be expected after above insertion
   System.out.println( "!!!!!!validation error: "+doc.GetErrorString() );
   System.out.println( "!!!!!!validation error: "+doc.GetErrorPos() );
   System.out.println( "!!!!!!validation error: "+doc.GetBadData() );

  }

  doc.ReleaseInstance();
  app.Quit();
  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*

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()
}
```

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 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 );
}
```

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 )
}
```

5.5 SpyDialogs

```
// Since version 2004R4
public class SpyDialogs
{
  public SpyApplication GetApplication();
  public SpyApplication GetParent();
  public SpyCodeGeneratorDlg GetCodeGeneratorDlg ();
  public SpyFileSelectionDlg GetFileSelectionDlg ();
  public SpySchemaDocumentationDlg GetSchemaDocumentationDlg ();
}
```

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();
  public int GetErrorPos();
  public SpyXMLData GetBadData();
  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 );
}
```

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 )
}
```

5.8 SpyElementList

```
public class SpyElementList
{
  public void ReleaseInstance();
  public long GetCount();
  public SpyElementListItem  GetItem( long nIndex );
  public void RemoveElement( long nIndex );
}
```

5.9 SpyElementListItem

```
public class SpyElementListItem
{
  public void ReleaseInstance();
  public long GetElementKind();
  public void SetElementKind( long nKind );
  public long GetFieldCount ();
  public String GetName();
  public long GetRecordCount();
}
```

5.10 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 );
}
```

5.11 SpyFileSelectionDlg

```
// Since version 2004R4
public class SpyFileSelectionDlg
{
  public SpyApplication GetApplication();
  public SpyDialogs GetParent();
  public String GetFullName();
  public void SetFullName( String i_strName );
  public long GetDialogAction();
  public void SetDialogAction( long i_eAction );
}
```

5.12 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 );
}
```

5.13 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 );
}
```

5.14 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 );
}
```

5.15 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 );
}
```

5.16 SpySchemaDocumentationDlg

```
// Since version 2004R4
public class SpySchemaDocumenttaionDlg
{
  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 );
  }
```

5.17 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 );
}
```

5.18 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 );
}
```

5.19 Authentic

5.19.1 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();
}
```

5.19.2 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 ();
}
```

5.19.3 SpyDocEditSelection

```
public class SpyDocEditSelection
{
  public void ReleaseInstance();
  public SpyXMLData GetEnd();
  public long GetEndTextPosition();
  public SpyXMLData GetStart();
  public long GetStartTextPosition();
}
```

5.19.4 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 );
}
```

5.20 Predefined constants

This section lists all classes that define the predefined constants used by the Java interface.

5.20.1 SPYAuthenticActions

```
public class SPYAuthenticActions
{
   public final static long                      = 0;
   spyAuthenticInsertAt
   public final static long spyAuthenticApply= 1;
   public final static long                      = 2;
   spyAuthenticClearSurr
   public final static long                      = 3;
   spyAuthenticAppend
   public final static long                      = 4;
   spyAuthenticInsertBefore
   public final static long                      = 5;
   spyAuthenticRemove
}
```

5.20.2 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
}
```

5.20.3 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
}
```

5.20.4 SPYAuthenticMarkupVisibility

```
public class SPYAuthenticMarkupVisibility
{
   public final static long                  = 0;
   spyAuthenticMarkupHidden
   public final static long                  = 1;
   spyAuthenticMarkupSmall
   public final static long                  = 2;
   spyAuthenticMarkupLarge
   public final static long                  = 3;
   spyAuthenticMarkupMixed
}
```

5.20.5 SPYDatabaseKind

```
public class SPYLoading
{
   public final static long           = 0;
   spyDB_Access
   public final static long           = 1;
   spyDB_SQLServer
   public final static long           = 2;
   spyDB_Oracle
   public final static long           = 3;
   spyDB_Sybase
   public final static long           = 4;
   spyDB_MySQL
   public final static long spyDB_DB2 = 5;
   public final static long           = 6;
   spyDB_Other
   public final static long           = 7;
   spyDB_Unspecified
}
```

5.20.6 SPYDialogAction

```
public class SPYDialogAction
{
   public final static long           = 0;
   spyDialogOK
   public final static long           = 1;
   spyDialogCancel
   public final static long           = 2;
   spyDialogUserInput
}
```

5.20.7 SPYDOMType

```
public class SPYDOMType
{
   public final static long           = 0;
   spyDOMType_msxml4
   public final static long           = 1;
   spyDOMType_xerces
}
```

5.20.8 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;
}
```

5.20.9 SPYEncodingByteOrder

```
public class SPYEncodingByteOrder
{
   public final static long spyNONE     = 0;
   public final static long             = 1;
   spyLITTLE_ENDIAN
   public final static long             = 2;
   spyBIG_ENDIAN
}
```

5.20.10 SPYExportNamespace

```
public class SPYExportNamespace
{
    public final static long spyNoNamespace                = 0;
    public final static long spyReplaceColonWithUnderscore = 1;
}
```

5.20.11 SPYFrequentElements

```
public class SPYFrequentElements
{
   public final static long         = 0;
   spyGlobalElements
   public final static long         = 1;
   spyGlobalComplexType
}
```

5.20.12 SPYLibType

```
public class SPYLibType
{
   public final static long          = 0;
   spyLibType_static
   public final static long          = 1;
   spyLibType_dll
}
```

5.20.13 SPYLoading

```
public class SPYLoading
{
   public final static long          = 0;
   spyUseCacheProxy
   public final static long          = 1;
   spyReload
}
```

5.20.14 SPYNameDateTimeFormat

```
public class  SPYNumberDateTimeFormat
{
   public final static long           = 0;
   spySystemLocale
   public final static long           = 1;
   spySchemaCompatible
}
```

5.20.15 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;
}
```

5.20.16 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
}
```

5.20.17 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
}
```

5.20.18 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                = 7;
   spyKindAttrGroup
   public final static long spyKindAttrAny = 8;
   public final static long                = 9;
   spyKindIdentityUnique
   public final static long                = 10;
   spyKindIdentityKey
   public final static long                = 11;
   spyKindIdentityKeyRef
   public final static long                = 12;
   spyKindIdentitySelector
   public final static long                = 13;
   spyKindIdentityField
   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;
}
```

5.20.19 SPYSchemaDocumentationFormat

```
public class SPYSchemaDocumentationFormat
{
   public final static long           = 0;
   spySchemaDoc_HTML
   public final static long           = 1;
   spySchemaDoc_MSWord
}
```

5.20.20 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;
}
```

5.20.21 SPYTextEnclosing

```
public class SPYTextEnclosing
{
   public final static long       = 0;
   spyNoEnclosing
   public final static long       = 1;
   spySingleQuote
   public final static long       = 2;
   spyDoubleQuote
}
```

5.20.22 SPYTypeDetection

```
public class SPYTypeDetection
{
   public final static long          = 0;
   spyBestPossible
   public final static long          = 1;
   spyNumbersOnly
   public final static long          = 2;
   spyNoDetection
}
```

5.20.23 SPYURLTypes

```
public class SPYURLTypes
{
   public final static long        = (-1);
   spyURLTypeAuto
   public final static long        = 0;
   spyURLTypeXML
   public final static long        = 1;
   spyURLTypeDTD
}
```

5.20.24 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
}
```

5.20.25 SPYWhitespaceComparison

```
public class SPYWhitespaceComparison
{
   public final static long            = 0;
   spyCompareAsIs
   public final static long            = 1;
   spyCompareNormalized
   public final static long            = 2;
   spyStripAll
}
```

5.20.26 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
  }
```

Altova XMLSpy Professional Edition User Manual

Appendices

Appendices

These appendices contain technical information about XMLSpy and important licensing information. Each appendix contains sub-sections as given below:

Technical Data

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

License Information

- Electronic software distribution
- 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 &#160; (the decimal character reference for a non-breaking space) is not inserted as &nbsp; 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 child whitespace-only text node that occurs between two elements within an element of mixed content.) This stripping may have an effect on the value returned by the fn:position(), fn:last(), and fn:count() functions.

For any node selection that selects text nodes also, boundary-whitespace-only text nodes would typically also be included in the selection. However, since the XML Infoset used by the Altova engines has boundary-whitespace-only text nodes stripped from it, these nodes are not present in the XML Infoset. As a result, the size of the selection and the numbering of nodes in the selection will be different than that for a selection which included these text nodes. The fn:position(), fn:last(), and fn:count() functions, therefore, could produce results that are different from those produced by some other processors.

A situation in which boundary-whitespace-only text nodes are evaluated as siblings of other elements arises most commonly when xsl:apply-templates is used to apply templates. When the fn:position(), fn:last(), and fn:count() functions are used in patterns with a name test (for example, para[3], which is short for para[position()=3]), boundary-whitespace-only nodes are irrelevant since only the named elements (para in the above example) are selected. (Note, however, that boundary-whitespace-only nodes **are** relevant in patterns that use the wildcard, for example, *[10].)

Note: If a boundary-whitespace-only text node is required in the output, then insert the required whitespace within one of the two adjoining child elements. For example, the XML fragment:

```
<para>This is <b>bold</b> <i>italic</>.</para>
```

when processed with the XSLT template

```
<xsl:template match="para">
   <xsl:apply-templates/>
</xsl:template>
```

will produce:

```
This is bolditalic.
```

To get a space between `bold` and `italic` in the output, insert a space character within either the `<b>` or `<i>` elements in the XML source. For example:

```
<para>This is <b>bold</b> <i> italic</>.</para> or
<para>This is <b>bold&#x20;</b> <i>italic</>.</para> or
<para>This is <b>bold</b><i>&#x20;italic</>.</para>
```

When such an XML fragment is processed with the same XSLT template given above, it will produce:

```
This is bold italic.
```

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 Candidate Recommendation of 8 June 2006. Note the following general information about the engine.

Backwards Compatibility
The Altova XSLT 2.0 Engine is not backwards compatible. Depending on the Altova product you are using, the following options are available:

- If you wish to run an XSLT 1.0 transformation using AltovaXML, then you should use the Altova XSLT 1.0 Engine of this package.
- If you are running an XSLT transformation, or carrying out an action involving an XSLT transformation, within the XMLSpy, StyleVision, Authentic, or MapForce products, the correct built-in Altova XSLT Engine (1.0 or 2.0) is automatically selected by the application. This selection is based on the value of the `version` attribute of the `stylesheet` or `transform` element of the stylesheet.

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

Namespace Name	Prefix	Namespace URI
XML Schema types	`xs:`	`http://www.w3.org/2001/XMLSchema`
XPath 2.0 functions	`fn:`	`http://www.w3.org/2005/xpath-functions`

Typically, these namespaces will be declared on the `xsl:stylesheet` or `xsl:transform` element, as shown in the following listing:

```
<xsl:stylesheet version="2.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:fn="http://www.w3.org/2005/xpath-functions"
   ...
</xsl:stylesheet>
```

The following points should be noted:

- The Altova XSLT 2.0 Engine uses the XPath 2.0 and XQuery 1.0 Functions namespace (listed in the table above) as its **default functions namespace**. So you can use XPath 2.0 and XSLT 2.0 functions in your stylesheet without any prefix. If you declare the

XPath 2.0 Functions namespace in your stylesheet with a prefix, then you can additionally use the prefix assigned in the declaration.

- When using type constructors and types from the XML Schema namespace, the prefix used in the namespace declaration must be used when calling the type constructor (for example, `xs:date`).
- With the CRs of 8 June 2006, the `untypedAtomic` and duration datatypes (`dayTimeDuration` and `yearMonthDuration`), which were formerly in the XPath Datatypes namespace (typically prefixed `xdt:`) have been moved to the XML Schema namespace.
- Some XPath 2.0 functions have the same name as XML Schema datatypes. For example, for the XPath functions `fn:string` and `fn:boolean` there exist XML Schema datatypes with the same local names: `xs:string` and `xs:boolean`. So if you were to use the XPath expression `string('Hello')`, the expression evaluates as `fn:string('Hello')`—not as `xs:string('Hello')`.

Schema-awareness

The Altova XSLT 2.0 Engine is schema-aware.

Whitespace in XML document

By default, the Altova XSLT 2.0 Engine strips all boundary whitespace from boundary-whitespace-only nodes in the source XML document. The removal of this whitespace affects the values that the `fn:position()`, `fn:last()`, `fn:count()`, and `fn:deep-equal()` functions return. For more details, see Whitespace-only Nodes in XML Document in the XPath 2.0 and XQuery 1.0 Functions section.

Note: If a boundary-whitespace-only text node is required in the output, then insert the required whitespace within one of the two adjoining child elements. For example, the XML fragment:

```
<para>This is <b>bold</b> <i>italic</>.</para>
```

when processed with the XSLT template

```
<xsl:template match="para">
   <xsl:apply-templates/>
</xsl:template>
```

will produce:

```
This is bolditalic.
```

To get a space between `bold` and `italic` in the output, insert a space character within either the `<b>` or `<i>` elements in the XML source. For example:

```
<para>This is <b>bold</b> <i> italic</>.</para> or
<para>This is <b>bold&#x20;</b> <i>italic</>.</para> or
<para>This is <b>bold</b><i>&#x20;italic</>.</para>
```

When such an XML fragment is processed with the same XSLT template given above, it will produce:

```
This is bold italic.
```

XSLT 2.0 elements and functions

Limitations and implementation-specific behavior of XSLT 2.0 elements and functions are listed in the section XSLT 2.0 Elements and Functions.

XPath 2.0 functions
Implementation-specific behavior of XPath 2.0 functions is listed in the section XPath 2.0 and XQuery 1.0 Functions.

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.

`format-date, format-dateTime, formatTime`
Presentation modifiers and formatting tokens in the variable markers of the Picture argument are not supported and, if supplied, are ignored. The optional Language, Calendar, and Country arguments are not supported and, if supplied, are ignored. Days and weeks are returned as numbers; in the case of single digit numbers, there is no preceding zero. The component specifier `F` is returned as a number. Weeks of the month are reckoned from Monday to Friday. The component specifier `P` returns `am` or `pm` (in English).

`function-available`
The function tests for the availability of XSLT 2.0 functions, not for the availability of XPath 2.0 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.

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 Candidate Recommendation of 8 June 2006. 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 8 June 2006, 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 CR of 8 June 2006.

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 Candidate Recommendation of 8 June 2006. The Altova XQuery 1.0 Engine implements the World Wide Web Consortium's (W3C's) XQuery 1.0 Candidate Recommendation of 8 June 2006. 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 CR of 8 June 2006.
- The Altova XPath 2.0 Engine conforms to the rules of XML 1.0 (Third 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 `base-uri()` function, it is still the base URI of the including XML document that is used—not the base URI of the external entity. • The use of the `xml:base` attribute in the source XML document is not supported. This means that the base URI of a node in the XML document cannot be altered using the `xml:base` attribute.
`collection`	• The `collection()` function is a mapping of form `(string, nodes)`, currently called `available collections` and left empty by the external environment. The function therefore returns either (i) the empty sequence (when called with no argument or with an empty sequence), or (ii) an error (when called with a non-empty argument).
`count`	• See note on whitespace in the General Information section.

contd./

Function Name	**Notes**

`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 ASCII character set only is supported.

`normalize-unicode`	• Not supported.
`position`	• See note on whitespace in the General Information section.

contd./

Function Name	Notes
`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 ASCII character set only is supported.

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.

My SQL 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 (constraint applied)	xs:boolean
NUMBER	xs:decimal
FLOAT	xs:double
DATE	xs:dateTime
INTERVAL YEAR TO MONTH	xs:gYearMonth
BLOB	xs:base64Binary

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	*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	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 NVARHCAR2
	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
This software application is a 32-bit Windows application that runs on Windows NT 4.0, Windows 2000, and Windows XP.

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 exponentially with the size of the document. For example, a 512kB document would typically require about 2MB of RAM, whereas a 5MB document can consume up to 50MB. Memory requirements are also influenced by the unlimited Undo history. When repeatedly cutting and pasting large selections in large documents, 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 new 16-bit character-set standard defined by the Unicode Consortium that 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, JustSystem, 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 NT4.0/2000/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.

Windows NT typically includes support for all common single-byte writing-systems in its Arial, Times, and Courier New fonts and will additionally include all required fonts for the writing-system in your own country (i.e. if you install the Japanese version of Windows NT you will automatically have fonts that support the Katakana, Hiragana, and Kanji writing-systems as well as the input-methods and dictionaries to enter Kanji and to switch between Katakana and Hiragana). If you wish to edit any document from a foreign writing-system, you may want to install additional Windows NT components for that writing-system or purchase special Unicode fonts for these writing-systems (such fonts are available from all leading type vendors).

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 | Registration**), 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 Open URL... dialog box to open a document directly from a URL (**File | Open 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.

Note: All Internet communication is initiated only when directly requested by you. from you! (This functionality is important in an XML development environment since XML is, after all, a technology closely related to the Internet.)

5 License Information

This section contains:

- Information about the distribution of this software product
- Information about the copyrights 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 End User License Agreement, which is delivered in the form of a key-code that you enter into the Registration dialog to unlock the product. You can purchase your license at the online shop at the Altova website.

Distributing the product
If you wish to share the product with others, please make sure that you distribute only the installation program, which is a convenient package that will install the application together with all sample files and the onscreen help. Any person that receives the product from you is also automatically entitled to a 30-day evaluation period. After the expiration of this period, any other user must also purchase a license in order to be able to continue using the product.

For further details, please refer to the End User License Agreement at the end of this section.

5.2 License Metering

Your Altova product has a built-in license metering module that 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, it 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. Other than that, it will not attempt to communicate over a network. If you are not connected to a LAN, or are using dial-up connections to connect to the Internet, the application will not generate any network traffic at all.

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 ensure 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..

Please note that your Altova product at no time attempts to send any information out of your LAN or over the Internet. 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* http://www.isi.edu/in-notes/iana/assignments/port-numbers *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 Copyright

All title and copyrights in this software product (including but not limited to images, photographs, animations, video, audio, music, text, and applets incorporated in the product), in the accompanying printed materials, and in any copies of these printed materials are owned by Altova GmbH or the respective supplier. This software product is protected by copyright laws and international treaty provisions.

- This software product ©1998-2006 Altova GmbH. All rights reserved.
- The Sentry Spelling-Checker Engine © 2000 Wintertree Software Inc.
- STLport © 1999, 2000 Boris Fomitchev, © 1994 Hewlett-Packard Company, © 1996, 1997 Silicon Graphics Computer Systems, Inc, © 1997 Moscow Center for SPARC Technology.
- Scintilla © 1998–2002 Neil Hodgson <`neilh@scintilla.org`>.
- "ANTLR Copyright © 1989-2005 by Terence Parr (www.antlr.org)"

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

Licensor:

Altova GmbH
Rudolfsplatz 13a/9
A-1010 Wien
Austria

Important - Read Carefully. Notice to User:

This End User License Agreement ("Software License Agreement") is a legal document between you and Altova GmbH ("Altova"). It is important that you read this document before using the Altova-provided software ("Software") and any accompanying documentation, including, without limitation printed materials, 'online' files, or electronic documentation ("Documentation"). By clicking the "I accept" and "Next" buttons below, or by installing, or otherwise using the Software, you agree to be bound by the terms of this Software License Agreement as well as the Altova Privacy Policy ("Privacy Policy") including, without limitation, the warranty disclaimers, limitation of liability, data use and termination provisions below, whether or not you decide to purchase the Software. You agree that this agreement is enforceable like any written agreement negotiated and signed by you. If you do not agree, you are not licensed to use the Software, and you must destroy any downloaded copies of the Software in your possession or control. Please 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.** Upon your acceptance of this Software License Agreement Altova grants you a non-exclusive, non-transferable (except as provided below), limited license to install and use a copy of the Software on your compatible computer, up to the Permitted Number of computers. The Permitted Number of computers shall be delineated at such time as you elect to purchase the Software. During the evaluation period, hereinafter defined, only a single user may install and use the software on one computer. 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. 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. In addition, if you have licensed XMLSpy Enterprise Edition or MapForce Enterprise Edition, or UModel, 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 into executable form the complete generated code comprised of the combination of the Restricted Source Code and the Unrestricted Source Code, and to use, copy, distribute or license that executable. You may not distribute or redistribute, sublicense, sell, or transfer to a third party the Restricted Source Code, unless said third party already has a license to the Restricted Source Code through their separate license agreement with Altova or other agreement with

Altova. 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 permitted in Section 1(h) reverse engineering of the Software is strictly prohibited as further detailed therein.

(b) **Server Use.** You may install one copy of the Software on your computer file server for the purpose of downloading and installing the Software onto other computers within your internal network up to the Permitted Number of computers. 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) **Concurrent Use**. If you have licensed a "Concurrent-User" version of the Software, you may install the Software on any compatible computers, 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. The Permitted Number of concurrent users shall be delineated at such time as you elect to purchase the Software licenses.

(d) **Backup and Archival Copies.** You may make one backup and one 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.

(e) **Home Use.** You, as the primary user of the computer on which the Software is installed, may also install the Software on one of your home computers for your use. However, the Software may not be used on your home computer at the same time as the Software is being used on the primary computer.

(f) **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. 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 update 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 update in addition to the Software that it is replacing. You agree that use of the upgrade of update terminates your license to use the Software or portion thereof replaced.

(g) **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.

(h) **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.

(i) **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 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. you agree that you are solely responsible for the accuracy and adequacy of the software for your intended use and you will indemnify and HOLD harmless ALTOVA from any 3rd party suit to the extent based upon the accuracy and adequacy of the software in your use. without limitation, The Software is not intended for use in the operation of nuclear facilities, aircraft navigation, communication systems or air traffic control equipment, where the failure of the Software could lead to death, personal injury or severe physical or environmental damage.

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, Markup Your Mind, Axad, Nanonull, and Altova are trademarks of Altova GmbH (registered in numerous countries). Unicode and the Unicode Logo are trademarks of Unicode, Inc. Windows, Windows 95, Windows 98, Windows NT, Windows 2000 and Windows XP 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. 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 with 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 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. 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 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.**
(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 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 as revised from time to time. European users understand and consent to the processing of personal information in the United States for the purposes described herein. 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.

8. TERM AND TERMINATION

This Software License Agreement may be terminated (a) by your giving Altova written notice of termination; or (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. 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 it 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(g), (h), (i), 2, 5(b), (c), 9, and 10 survive termination as applicable.

9. RESTRICTED RIGHTS NOTICE AND EXPORT RESTRICTIONS

The Software was developed entirely at private expense and is commercial computer software provided with **RESTRICTED RIGHTS**. Use, duplication or disclosure by the U.S. Government or a U.S. Government contractor or subcontractor is subject to the restrictions set forth in this Agreement and as provided in FAR 12.211 and 12.212 (48 C.F.R. §12.211 and 12.212) or DFARS 227. 7202 (48 C.F.R. §227-7202) as applicable. Consistent with the above as applicable, Commercial Computer Software and Commercial Computer Documentation licensed to U.S. government end users only as commercial items and only with those rights as are granted to all other end users under the terms and conditions set forth in this Software License Agreement. Manufacturer is Altova GmbH, Rudolfsplatz, 13a/9, A-1010 Vienna, Austria/EU. You may not use or otherwise export or re-export the Software or Documentation except as authorized by United States law and the laws of the jurisdiction in which the Software was obtained. In particular, but without limitation, the Software or Documentation may not be exported or re-exported (i) into (or to a national or resident of) any U.S. embargoed country or (ii) to anyone on the U.S. Treasury Department's list of Specially Designated Nationals or the U.S. Department of Commerce's Table of Denial Orders. By using the Software, you represent and warrant that you are not located in, under control of, or a national or resident of any such country or on any such list.

10. 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 Massachusetts and you further agree and expressly consent to the exercise of personal jurisdiction in the federal or state courts 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: 2005-05-05

Index