

# **AltovaXML 2012**

## **User and Reference Manual**

## **AltovaXML 2012 User & Reference Manual**

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

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

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

Published: 2012

© 2012 Altova GmbH

---

# Table of Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Product Features .....	4
1.2	Available Functionality .....	5
1.3	System Requirements and Installation .....	6
1.4	About this Documentation .....	7
<b>2</b>	<b>Usage</b>	<b>10</b>
2.1	Command Line .....	11
2.1.1	XML Validation and Well-Formedness .....	13
2.1.2	XSLT 1.0 Transformations .....	16
2.1.3	XSLT 2.0 Transformations .....	18
2.1.4	XQuery 1.0 Executions .....	20
2.2	COM Interface .....	22
2.2.1	Registering AltovaXML as a COM Server Object .....	23
2.2.2	AltovaXML Object Model .....	24
2.2.3	Application .....	25
2.2.4	XMLValidator .....	26
2.2.5	XSLT1 .....	28
2.2.6	XSLT2 .....	30
2.2.7	XQuery .....	33
2.2.8	Examples .....	36
	<i>Visual Basic</i> .....	36
	<i>JScript</i> .....	37
	<i>C++</i> .....	38
2.3	Java Interface .....	40
2.3.1	Example Java Project .....	41
2.3.2	Classes .....	44
	<i>AltovaXMLFactory</i> .....	44
	<i>XMLValidator</i> .....	45
	<i>XQuery</i> .....	47
	<i>XSLT1</i> .....	50
	<i>XSLT2</i> .....	52
2.3.3	Old Java API (Obsolete) .....	55

	<i>Example</i> .....	56
	<i>Interfaces</i> .....	58
	.....IAltovaXMLEngine.....	58
	.....IAltovaXMLFactory.....	59
	.....IExecutable.....	60
	.....IReleasable.....	60
	.....IXMLValidator.....	61
	.....IXQuery.....	62
	.....IXSLT.....	64
	<i>Classes</i> .....	65
	.....AltovaXMLFactory.....	65
	.....XMLValidator.....	67
	.....XQuery.....	69
	.....XSLT1.....	73
	.....XSLT2.....	76
2.4	.NET Interface .....	80
2.4.1	General Usage .....	82
2.4.2	Altova.AltovaXML.XMLValidator .....	84
2.4.3	Altova.AltovaXML.XSLT1 .....	86
2.4.4	Altova.AltovaXML.XSLT2 .....	91
2.4.5	Altova.AltovaXML.XQuery .....	96
2.4.6	Example .....	100
2.5	Explicitly Releasing AltovaXML COM-Server from C# and VB.NET .....	102
2.6	OOXML and ZIP Files .....	103

### **3 Engine Information 106**

3.1	Altova XML Validator .....	107
3.2	XSLT 1.0 Engine: Implementation Information .....	108
3.3	XSLT 2.0 Engine: Implementation Information .....	110
3.3.1	General Information .....	111
3.3.2	XSLT 2.0 Elements and Functions .....	113
3.4	XQuery 1.0 Engine: Implementation Information .....	114
3.5	XPath 2.0 and XQuery 1.0 Functions .....	117
3.5.1	General Information .....	118
3.5.2	Functions Support .....	120
3.6	Extensions .....	123
3.6.1	Java Extension Functions .....	124
	<i>User-Defined Class Files</i> .....	125
	<i>User-Defined Jar Files</i> .....	127
	<i>Java: Constructors</i> .....	128
	<i>Java: Static Methods and Static Fields</i> .....	128
	<i>Java: Instance Methods and Instance Fields</i> .....	129

---

	<i>Datatypes: XPath/XQuery to Java</i> .....	130
	<i>Datatypes: Java to XPath/XQuery</i> .....	131
3.6.2	.NET Extension Functions .....	132
	<i>.NET: Constructors</i> .....	134
	<i>.NET: Static Methods and Static Fields</i> .....	134
	<i>.NET: Instance Methods and Instance Fields</i> .....	135
	<i>Datatypes: XPath/XQuery to .NET</i> .....	136
	<i>Datatypes: .NET to XPath/XQuery</i> .....	137
3.6.3	MSXSL Scripts for XSLT .....	138
3.6.4	Altova Extension Functions .....	141
	<i>General Functions</i> .....	141
	<i>Barcode Functions</i> .....	144
	<i>Chart Functions</i> .....	146
	<i>Chart Data XML Structure</i> .....	150
	<i>Example: Chart Functions</i> .....	154

## **4 License Information 160**

4.1	Electronic Software Distribution .....	161
4.2	Software Activation and License Metering .....	162
4.3	Intellectual Property Rights .....	163
4.4	License Agreement .....	164

## **Index**



## **Chapter 1**

---

### **Introduction**





# 1 Introduction

**AltovaXML 2012 Reporting Edition** is an XML application package which contains the Altova XML Validator, Altova XSLT 1.0 Engine, Altova XSLT 2.0 Engine, and Altova XQuery 1.0 Engine. The package is available, free of charge, as a single installer file from the [Altova website](#). AltovaXML can be used to validate XML documents, transform XML documents using XSLT stylesheets, and execute XQuery documents.

AltovaXML can be used from the command line, via a COM interface, in Java programs, and in .NET applications. This documentation describes the usage of AltovaXML in all these environments, and also lists implementation-specific aspects of the engines in the package.

Last updated: 02/17/2012

## 1.1 Product Features

The main features of AltovaXML are as follows:

### Package

- XML Validator, XSLT Engines, and XQuery Engine packaged as a single installer file.
- Installer file available for download from [Altova website](#) free-of-charge.
- Easy installation of executable files on Windows systems.

### Command line

- Command line usage for validation, XSLT transformation, and XQuery execution.
- Validation of XML documents according to DTD and W3C XML Schema rules.
- Transformation of XML documents with XSLT 1.0 and XSLT 2.0 stylesheets in conformance with respective W3C specifications.
- Execution of XQuery 1.0 documents in conformance with W3C specifications.

### COM interface

- Can be used via COM interface, and therefore with applications and scripting languages that support COM.
- COM interface support is implemented for Raw and Dispatch interfaces.
- Wide range of XML validation, XSLT transformation, and XQuery execution features are available through interface properties.
- XML, DTD, XML Schema, XSLT, and XQuery input can be provided as files or as text strings in scripts and in application data.

### Java interface

- AltovaXML functionality is available as Java classes that can be used in Java programs.
- Java classes provide XML validation, XSLT transformation, and XQuery execution features.

### .NET interface

- A DLL file is built as a wrapper around AltovaXML and allows .NET users to connect to the functionality of AltovaXML.
- Provides primary interop assembly signed by Altova.
- Wide range of XML validation, XSLT transformation, and XQuery execution features are available.
- XML, DTD, XML Schema, XSLT, and XQuery input can be provided as files or as text strings in scripts and in application data.

## 1.2 Available Functionality

AltovaXML provides the functionality listed below. Most of this functionality is common to command line usage and COM interface usage. One major difference is that COM interface usage allows documents to be constructed from text strings via the application or scripting code (instead of referencing XML, DTD, XML Schema, XSLT, or XQuery files).

### XML and XBRL Validation

- Validates the supplied XML document, returning valid or invalid.
- Validation can be done against the DTD or XML Schema referenced within the XML file, or against an external DTD or XML Schema supplied by a command line parameter or a COM interface property.
- Checks well-formedness of the supplied XML document, separately from validation.
- Validates XBRL documents. The XBRL document is validated against an XBRL taxonomy (which is a `.xsd` file) according to the rules of XBRL.

### XSLT Transformations

- Transforms supplied XML document using supplied XSLT 1.0 or XSLT 2.0 document.
- XML document can be provided as a file via the input of a URL. In the case of usage via the COM interface, the XML document can alternatively be supplied as a text string.
- XSLT document can be provided as a file via the input of a URL. In the case of usage via the COM interface, the XSLT document can alternatively be supplied as a text string.
- Returns output documents at the named location. When called via COM interface can also return output documents as a string.
- XSLT parameters can be supplied via the command line and via the COM interface.
- Altova extension functions (including, in the Reporting Edition, for charts) enable specialized processing.

### XQuery Execution

- Executes the supplied XQuery 1.0 document, optionally against an XML document named in a command line parameter or a COM interface property.
- XQuery document can be provided as a file via the input of a URL. In the case of usage via the COM interface, the XQuery document can alternatively be supplied as a text string.
- XML document can be provided as a file via the input of a URL. In the case of usage via the COM interface, the XML document can alternatively be supplied as a text string.
- Returns output documents at the named location. When called via COM interface can also return output documents as a string.
- External XQuery variables can be supplied via the command line and via the COM interface.
- Serialization options include: output encoding, output method (that is, whether the output is XML, XHTML, HTML, or Text), omitting the XML declaration, and indentation.
- Altova extension functions (including, in the Reporting Edition, for charts) enable specialized processing.

## 1.3 System Requirements and Installation

### System requirements

AltovaXML is supported on Windows NT, Windows XP, Windows Server 2003, Windows Server 2008, Windows Vista, and Windows 7. To use AltovaXML via a COM interface, users should have privileges to use the COM interface, that is, to register the application and execute the relevant applications and/or scripts. AltovaXML Reporting Edition is available for both 32-bit and 64-bit machines. AltovaXML Community Edition is available for 32-bit machines only.

### Installation

AltovaXML is available on the [Altova website](#) as a self-extracting download that will install AltovaXML with the necessary registrations. After you have downloaded the installer file (`AltovaXML2012.exe`) to your machine, double-click it to start the installation. The installer will install AltovaXML in the `Altova/AltovaXML2012` folder in the `Program Files` folder. All the necessary registrations to use AltovaXML via a COM interface, as a Java interface, and in the .NET environment will be done by the installer. This includes registering the AltovaXML executable as a COM server object, installing `AltovaXMLLib.dll` (for Java interface usage) in the `WINDIR\system32\` directory, and adding the `Altova.AltovaXML.dll` file to the .NET reference library.

You should note the following:

- For command line usage, invoke the installed executable file (`AltovaXML.exe`). This file can be copied to another accessible location on your machine or network and invoked from there.
- You can straightaway use AltovaXML via COM interface since the installed executable file `AltovaXML_COM.exe` will have been registered as a COM server object. If you change the location of the executable file `AltovaXML_COM.exe` to another location on your machine or to a mapped network drive, then you must manually register it at its new location as a COM server object. How to do this described in the section, [Registering AltovaXML as a COM server object](#).
- In order to use AltovaXML via a Java interface, `AltovaXML_COM.exe` must be registered as a COM server object and the Java libraries must reside in the `classpath`. The Java libraries are installed in the folder: `JavaAPI` in the AltovaXML application folder. Registration as a COM server object is done automatically by the installer process. Note that, if you change the location of the file `AltovaXML_COM.exe` after installation, then you must manually register it at its new location as a COM server object. See [Registering AltovaXML as a COM Server Object](#) and [Java Interface](#) for details.

## 1.4 About this Documentation

This documentation is the official product documentation of AltovaXML and provides comprehensive information about it. Its structure is as follows:

- The [Introduction](#) describes the features of the AltovaXML product, the functionality it provides, the main system requirements to use AltovaXML, and how AltovaXML is to be installed.
- The [Usage](#) section describes how to use AltovaXML from the command line and via a COM interface. The [Command Line](#) section provides details about the syntax used to invoke the various functionalities of AltovaXML. The [COM Interface](#) section describes how AltovaXML can be used with a COM interface; it provides a detailed description of the object model, its interfaces, and the properties of interfaces. The [Java Interface](#) section describes how AltovaXML can be used with Java and lists the defined Java interfaces and classes. The [.NET Interface](#) section provides a description of usage and lists the various methods and properties that can be used.
- The [Engine Information](#) section describes implementation-specific aspects of the various engines that are components of AltovaXML. Each engine is described separately.



## Chapter 2

---

### Usage

## 2 Usage

After AltovaXML has been downloaded and installed at the desired location, you can use it in the following ways:

- By calling the application from the [command line](#),
- By using the application via a [COM interface](#),
- By using the application via a [Java interface](#), and
- By using the application in the [.NET environment](#).



## 2.1 Command Line

To use AltovaXML from the command line, the executable file (`AltovaXML.exe`) must be installed/copied to an accessible location on your machine or network. The general syntax to call the application is:

```
AltovaXML functionality arg1 ... argN [options]
```

where

<code>AltovaXML</code>	Calls the application.
<code>functionality</code>	Specifies whether the XML validation, well-formedness check, XSLT 1.0 transformation, XSLT 2.0 transformation, or XQuery 1.0 execution functionality is called. Respective values are <code>-validate</code> (or <code>-v</code> ), <code>-wellformed</code> (or <code>-w</code> ), <code>-xslt1</code> , <code>-xslt2</code> , <code>-xquery</code> (or <code>-xq</code> ).
<code>arg1 ... argN</code>	The arguments of the called functionality.
<code>options</code>	Each functionality has its own set of options. These are described in the corresponding sub-sections of this section.

### General options

<code>-help, -h, or -?</code>	Displays usage information, i.e. a list of all arguments and options.
<code>-version, -ver</code>	Displays the program version.

The following functionality is available, and the allowed arguments and options for each functionality are described in detail in the corresponding sections:

- [XML Validation and Well-Formedness](#)
- [XSLT 1.0 Transformations](#)
- [XSLT 2.0 Transformations](#)
- [XQuery 1.0 Executions](#)

### Usage summary

Given below is a summary of command line usage. For details, refer to the respective sections.

#### [Using Altova XML Validator](#)

- `-validate <filename> [-schema <filename> | -dtd <filename>]`
- `-wellformed <filename>`

#### [Using Altova XSLT 1.0 Engine](#)

- `-xslt1 <filename> -in <filename> [-param name=value] [-out <filename>]`

#### [Using Altova XSLT 2.0 Engine](#)

- `-xslt2 <filename> -in <filename> [-param name=value] [-out <filename>]`

#### [Using Altova XQuery 1.0 Engine](#)

- `-xquery <filename> [-in <filename>] [-param name=value] [-out <filename>] [serialization options]`

**Note:** If the filename or the path to it contains a space, then the entire path should be enclosed in quotes. For example: `"c:\My Files\MyXML.xml"` or `"c:\MyFiles\My XML.xml"`.

## 2.1.1 XML Validation and Well-Formedness

### XML Validation syntax

The syntax to invoke **XML validation** is:

```
AltovaXML -validate xmlfile [-schema schemafile | -dtd dtdfile]  
[ options]
```

where

AltovaXML	Calls the application
-validate (or -v)	Specifies that the Altova XML Validator is to be used to validate the file <i>xmlfile</i> .

The following options are available:

-schema (or -s)	Specifies the XML Schema file <i>schemafile</i> to be used for validation.
-dtd (or -d)	Specifies the DTD file <i>dtdfile</i> to be used for validation.
-xbrlConsistency (or -xc)	Checks the semantics of XBRL documents.

### Mapping options

The following mapping options are available using the XML Catalogs mechanism and Altova Global Resources mechanism. (Altova Global Resources can be properly used only if an Altova product that supports Altova Global Resources is installed, for example, Altova XMLSpy.)

-catalog (or -c) [ <filename>]	Activates catalog mapping using the catalog given. If no file is specified, a catalog named <i>RootCatalog.xml</i> in the AltovaXML application folder will be assumed as the default.
-globalresources (or -gr) [ <filename>]	Activates Altova global resource mapping using the given Global Resources XML file or, if no file is specified, <i>GlobalResources.xml</i> in <i>My Documents/Altova</i> .
-globalresourcecon fig (or -gc) [ <name>]	Sets the active global resource configuration.

### Note about Global Resources

There are two settings required to select a resource using the Altova Global Resources mechanism:

- The Global Resources XML File contains definitions of the global resources. This file can be specified with the `-globalresources` (or `-gr`) option. If no file is specified, then the file *GlobalResources.xml* in the *My Documents/Altova* folder will be used.
- Each global resource in the Global Resources XML File can have multiple configurations, with each configuration mapping to a resource. The `-globalresourceconfig` (or `-gc`) option enables you to specify which configuration to use, by extension specifying which resource to use.

**Note:**

- When no XML Schema or DTD file is specified as a command line option, an XML Schema or DTD file must be specified in the XML document itself.
- If an XML Schema or DTD file is specified as a command line option **and** an XML Schema or DTD file is referenced in the XML file, then the file specified in the command line option is used for validation.
- If an XBRL instance document is validated, the XBRL taxonomy, which is a `.xsd` file, is looked up. If semantic validation is required in addition to syntactic validity, use the `-xbrlConsistency` option.

### Well-formedness Check syntax

The syntax to invoke the **well-formedness check** is:

```
AltovaXML -wellformed xmlfile
```

where

AltovaXML	Calls the application
-wellformed (or -w)	Specifies that the Altova XML Validator is to be used to check the well-formedness of the file <code>xmlfile</code> .

### Mapping options

The following mapping options are available using the XML Catalogs mechanism and Altova Global Resources mechanism. (Altova Global Resources can be properly used only if an Altova product that supports Altova Global Resources is installed, for example, Altova XMLSpy.)

-catalog (or -c) [ <filename>]	Activates catalog mapping using the catalog given. If no file is specified, a catalog named <code>RootCatalog.xml</code> in the AltovaXML application folder will be assumed as the default.
-globalresources (or -gr) [ <filename>]	Activates Altova global resource mapping using the given Global Resources XML file or, if no file is specified, <code>GlobalResources.xml</code> in <code>My Documents/Altova</code> .
-globalresourceconfig (or -gc) [ <name>]	Sets the active global resource configuration.

### Note about Global Resources

There are two settings required to select a resource using the Altova Global Resources mechanism:

- The Global Resources XML File contains definitions of the global resources. This file can be specified with the `-globalresources` (or `-gr`) option. If no file is specified, then the file `GlobalResources.xml` in the `My Documents/Altova` folder will be used.
- Each global resource in the Global Resources XML File can have multiple configurations, with each configuration mapping to a resource. The `-globalresourceconfig` (or `-gc`) option enables you to specify which configuration to use, by extension specifying which resource to use.

### Examples

- `AltovaXML -validate test.xml -schema testschema.xsd`
- `AltovaXML -v test.xml -dtd testdtd.dtd`
- `AltovaXML -wellformed test.xml`
- `AltovaXML -w test.xml`
- `AltovaXML -v test.xml -dtd testdtd.dtd -c MyCatalog.xml`
- `AltovaXML -validate test.xml -schema testschema.xsd -xc`

**Note:** For using Altova XML in batch commands, it is important to know the following:

- The return code of the last executed command is stored in the `errorlevel` variable, the value of which can be retrieved with a batch command such as `ECHO %errorlevel%`.
- The return codes are 0 = well-formed/valid; 1 = not well-formed/invalid.

## 2.1.2 XSLT 1.0 Transformations

### Syntax

The syntax to invoke XSLT 1.0 transformations is:

```
AltovaXML -xslt1 xsltfile -in xmlfile [-out outputfile] [options]
```

where

<code>AltovaXML</code>	Calls the application.
<code>-xslt1</code>	Specifies that the Altova XSLT 1.0 Engine is to be used for an XSLT transformation; the engine uses the XSLT 1.0 file <code>xsltfile</code> for the transformation.
<code>-in</code>	Specifies the XML file <code>xmlfile</code> to be transformed and its location.
<code>-out</code>	Specifies the output file <code>outputfile</code> and its location. If this option is omitted, the output is written to standard output.

The following options are available:

<code>-param</code>	Takes the instruction <code>paramname=XPath expression</code> . The <code>-param</code> switch is used before each global parameter. Double quotes must be used if a space is included in an XPath expression—whether in a path expression itself or in a string literal in the expression. See examples.
<code>-xslstack</code>	The stack size is the maximum depth of executed instructions, and can be changed with the <code>-xslstack</code> value. The minimum allowed value is 100. The default stack size is 1000. If the stack size is exceeded during a transformation, an error is reported.
<code>-namedTemplate</code> (or <code>-n</code> )	Sets the initial named template. A space separates the argument from its value. Example: <code>-namedTemplate MyTemplate</code>
<code>-mode</code> (or <code>-m</code> )	Sets the initial template mode. A space separates the argument from its value. Example: <code>-mode MyMode</code>

### Mapping options

The following mapping options are available using the XML Catalogs mechanism and Altova Global Resources mechanism. (Altova Global Resources can be properly used only if an Altova product that supports Altova Global Resources is installed, for example, Altova XMLSpy.)

<code>-catalog</code> (or <code>-c</code> ) [ <filename>]	Activates catalog mapping using the catalog given. If no file is specified, a catalog named <code>RootCatalog.xml</code> in the AltovaXML application folder will be assumed as the default.
<code>-globalresources</code> (or <code>-gr</code> ) [ <filename>]	Activates Altova global resource mapping using the given Global Resources XML file or, if no file is specified, <code>GlobalResources.xml</code> in <code>My Documents/Altova</code> .
<code>-globalresourcecon</code> <code>fig</code> (or <code>-gc</code> ) [ <name>]	Sets the active global resource configuration.

### Note about Global Resources

There are two settings required to select a resource using the Altova Global Resources mechanism:

- The Global Resources XML File contains definitions of the global resources. This file can be specified with the `-globalresources` (or `-gr`) option. If no file is specified, then the file `GlobalResources.xml` in the `My Documents/Altova` folder will be used.
- Each global resource in the Global Resources XML File can have multiple configurations, with each configuration mapping to a resource. The `-globalresourceconfig` (or `-gc`) option enables you to specify which configuration to use, by extension specifying which resource to use.

**Note:**

- The XSLT file must be specified in the command line instruction; an XSLT file referenced in an `<?xml-stylesheet?>` processing instruction in the XML document is not automatically used.
- If the `-out` parameter is omitted, output is written to the standard output.

**Examples**

- `AltovaXML -xslt1 test.xslt -in test.xml -out testout.xml`
- `AltovaXML -xslt1 test.xslt -in test.xml -out testout.xml -c MyCatalog.xml`
- `AltovaXML -xslt1 test.xslt -in test.xml -out testout.xml -param date=//node/@att1`
- `AltovaXML -xslt1 test.xslt -in test.xml -out testout.xml -param date="//node/@att1 | //node/@att2"`
- `AltovaXML -xslt1 test.xslt -in test.xml -out testout.xml -param date=node/@att1 -param title='stringwithoutspace'`
- `AltovaXML -xslt1 test.xslt -in test.xml -out testout.xml -param date=node/@att1 -param title="'string with spaces' "`

### 2.1.3 XSLT 2.0 Transformations

#### Syntax

The syntax to invoke XSLT 2.0 transformations is:

```
AltovaXML -xslt2 xsltfile -in xmlfile [-out outputfile] [options]
```

where

<code>AltovaXML</code>	Calls the application.
<code>-xslt2</code>	Specifies that the Altova XSLT 2.0 Engine is to be used for an XSLT transformation; the engine uses the XSLT 2.0 file <code>xsltfile</code> for the transformation.
<code>-in</code>	Specifies the XML file <code>xmlfile</code> to be transformed and its location.
<code>-out</code>	Specifies the output file <code>outputfile</code> and its location. If this option is omitted, the output is written to standard output.

The following options are available:

<code>-param</code>	Takes the instruction <code>paramname=XPath expression</code> . The <code>-param</code> switch is used before each global parameter. Double quotes must be used if a space is included in an XPath expression—whether in a path expression itself or in a string literal in the expression. See examples.
<code>-xslstack</code>	The stack size is the maximum depth of executed instructions, and can be changed with the <code>-xslstack</code> value. The minimum allowed value is 100. The default stack size is 1000. If the stack size is exceeded during a transformation, an error is reported.
<code>-namedTemplate</code> (or <code>-n</code> )	Sets the initial named template. A space separates the argument from its value. Example: <code>-namedTemplate MyTemplate</code>
<code>-mode</code> (or <code>-m</code> )	Sets the initial template mode. A space separates the argument from its value. Example: <code>-mode MyMode</code>

#### Mapping options

The following mapping options are available using the XML Catalogs mechanism and Altova Global Resources mechanism. (Altova Global Resources can be properly used only if an Altova product that supports Altova Global Resources is installed, for example, Altova XMLSpy.)

<code>-catalog</code> (or <code>-c</code> ) [ <filename>]	Activates catalog mapping using the catalog given. If no file is specified, a catalog named <code>RootCatalog.xml</code> in the AltovaXML application folder will be assumed as the default.
<code>-globalresources</code> (or <code>-gr</code> ) [ <filename>]	Activates Altova global resource mapping using the given Global Resources XML file or, if no file is specified, <code>GlobalResources.xml</code> in <code>My Documents/Altova</code> .
<code>-globalresourcecon</code> <code>fig</code> (or <code>-gc</code> ) [ <name>]	Sets the active global resource configuration.

#### Note about Global Resources

There are two settings required to select a resource using the Altova Global Resources mechanism:



- The Global Resources XML File contains definitions of the global resources. This file can be specified with the `-globalresources` (or `-gr`) option. If no file is specified, then the file `GlobalResources.xml` in the `My Documents/Altova` folder will be used.
- Each global resource in the Global Resources XML File can have multiple configurations, with each configuration mapping to a resource. The `-globalresourceconfig` (or `-gc`) option enables you to specify which configuration to use, by extension specifying which resource to use.

**Note:**

- The XSLT file must be specified in the command line instruction; an XSLT file referenced in an `<?xml-stylesheet?>` processing instruction in the XML document is not automatically used.
- If the `-out` parameter is omitted, output is written to the standard output.
- The XSLT 2.0 Engine can be used in its backward compatibility mode to process an XSLT 1.0 stylesheet. The output, however, could be different than that produced by the XSLT 1.0 Engine processing the same XSLT 1.0 stylesheet.

**Examples**

- `AltovaXML -xslt2 test.xslt -in test.xml -out testout.xml`
- `AltovaXML -xslt2 test.xslt -in test.xml -out testout.xml -c MyCatalog.xml`
- `AltovaXML -xslt2 test.xslt -in test.xml -out testout.xml -param date="//node/@att1`
- `AltovaXML -xslt2 test.xslt -in test.xml -out testout.xml -param date="//node/@att1 | //node/@att2"`
- `AltovaXML -xslt2 test.xslt -in test.xml -out testout.xml -param date=node/@att1 -param title='stringwithoutspace'`
- `AltovaXML -xslt2 test.xslt -in test.xml -out testout.xml -param date=node/@att1 -param title="'string with spaces' "`

## 2.1.4 XQuery 1.0 Executions

### Syntax

The syntax to invoke XQuery 1.0 executions is:

```
AltovaXML -xquery xqueryfile [-in inputXMLfile -out outputfile]
  [options]
```

where

<code>AltovaXML</code>	Calls the application.
<code>-xquery</code> (or <code>-xq</code> )	Specifies that the Altova XQuery 1.0 Engine is to be used for an XQuery execution of the file <code>xqueryfile</code> .
<code>-in</code>	Specifies the input XML file.
<code>-out</code>	Specifies the output file and its location. If this option is omitted, output is written to the standard output.

The following options are available:

<code>-var</code>	Specifies an external variable and its value. Takes the form <code>name=value</code> . Any number of external variables can be submitted, but each must be preceded by the <code>-var</code> keyword. Variable values must be strings that conform to the lexical form of the datatype as which the variable has been declared.
<code>-xparam</code>	Specifies an XQuery parameter name and the parameter's value. Takes the form <code>name=XPathExpression</code> . Use double quotes to enclose the XPath expression if the expression contains spaces. Use single quotes to delimit string literals in the XPath expression. Any number of parameters can be submitted, but each must be preceded by the <code>-xparam</code> keyword.
<code>-outputMethod</code> (or <code>-om</code> )	Serialization option to specify the type of output. Valid values are <code>xml</code> , <code>html</code> , <code>xhtml</code> , and <code>text</code> . Default is <code>xml</code> .
<code>-omitXMLDeclaration</code> (or <code>-od</code> )	Serialization option to specify whether the XML declaration should be omitted from the output or not. Valid values are <code>yes</code> and <code>no</code> . Default is <code>yes</code> .
<code>-outputIndent</code> (or <code>-oi</code> )	Serialization option to specify whether the output should be indented or not. Valid values are <code>yes</code> and <code>no</code> . Default is <code>no</code> .
<code>-outputEncoding</code> (or <code>-oe</code> )	Serialization option to specify the character set of the output. Valid values are names in the IANA character set registry. Default is <code>UTF-8</code> .

### Mapping options

The following mapping options are available using the XML Catalogs mechanism and Altova Global Resources mechanism. (Altova Global Resources can be properly used only if an Altova product that supports Altova Global Resources is installed, for example, Altova XMLSpy.)

<code>-catalog</code> (or <code>-c</code> ) [ <filename>]	Activates catalog mapping using the catalog given. If no file is specified, a catalog named <code>RootCatalog.xml</code> in the AltovaXML application folder will be assumed as the default.
--	--

```
-globalresources (or -gr) [ <filename>] Activates Altova global resource mapping using the given Global Resources XML file or, if no file is specified, GlobalResources.xml in My Documents/Altova.
-globalresourceconfig (or -gc) [ <name>] Sets the active global resource configuration.
```

### Note about Global Resources

There are two settings required to select a resource using the Altova Global Resources mechanism:

- The Global Resources XML File contains definitions of the global resources. This file can be specified with the `-globalresources (or -gr)` option. If no file is specified, then the file `GlobalResources.xml` in the `My Documents/Altova` folder will be used.
- Each global resource in the Global Resources XML File can have multiple configurations, with each configuration mapping to a resource. The `-globalresourceconfig (or -gc)` option enables you to specify which configuration to use, by extension specifying which resource to use.

**Note:** If the `-out` parameter is omitted, output is written to the standard output.

### Examples

- `AltovaXML -xquery testquery.xq -out testout.xml`
- `AltovaXML -xquery testquery.xq -in products.xml -out testout.xml`  
`-var company=Altova -var date=2006-01-01`
- `AltovaXML -xquery testquery.xq -out testout.xml`  
`-xparam source=" doc( 'c:\test\books.xml' )//book "`
- `AltovaXML -xquery testquery.xq -in products.xml -out testout.xml`  
`-var company=Altova -omitXMLDeclaration no -oe ASCII`

## 2.2 COM Interface

When registered as a COM server object, AltovaXML can be invoked from within applications and scripting languages that have programming support for COM calls. This is useful because it enables XML document validation, XSLT transformations (XSLT 1.0 and XSLT 2.0), and XQuery 1.0 document executions to be performed, by AltovaXML, from within a wide range of user applications.

To use AltovaXML with applications and scripting languages that have a COM interface, you must first register AltovaXML as a COM server object. How to do this is described in [Registering AltovaXML as a COM server object](#).

The AltovaXML object model and its properties are described in the following sub-sections of this section. (Note that you can use both the Raw Interface and Dispatch Interface of COM. The Raw Interface is used for programming languages (such as C++). The Dispatch Interface is used for scripting languages (such as JavaScript) that do not allow passing parameters by reference.) You can therefore use AltovaXML with:

- Scripting languages such as JavaScript or any other scripting language that supports the COM interface.
- Programming languages such as C++ or any other that supports the COM interface.
- Java and .NET, for which interfaces are built as a wrapper, with classes being created around the COM interface.

This section on COM interface usage ends with a set of examples of how various functionalities of AltovaXML can be invoked from within a variety of user applications.

### Examples

For examples additional to those in this section, see the example files in the `Examples` folder in the application folder.

## 2.2.1 Registering AltovaXML as a COM Server Object

When you install AltovaXML 2012, `AltovaXML_COM.exe` will automatically be registered as a COM server object. If you need to change the location of `AltovaXML_COM.exe`, it is best to de-install AltovaXML and then re-install it at the required location. In this way the necessary unregistration and registration are carried out by the installer process. If you copy `AltovaXML_COM.exe` to another machine, you must manually register AltovaXML at its new location as a COM server object. How to do this is explained below. This description assumes that AltovaXML has been successfully installed.

### Manual registration

To register AltovaXML as a COM server object, do the following:

1. Copy `AltovaXML_COM.exe` to the required location. If this location is not on the local machine, map this location to a network folder.
2. Open a Windows Command Prompt window, or, from the Start menu, select **Run...**
3. Register the application as a COM server object by using the `/regserver` parameter. For example, if `AltovaXML_COM.exe` is in the folder `c:\AltovaXML`, then key in:

```
c:\AltovaXML\AltovaXML_COM.exe /regserver
```

and press **Enter**.

### Checking success of the registration

If the registration was successful, the Registry should contain the classes `AltovaXML.Application` and `AltovaXML.Application.1`. These two classes will typically be found under `HKEY_LOCAL_MACHINE\SOFTWARE\Classes`.

### Manual unregistration

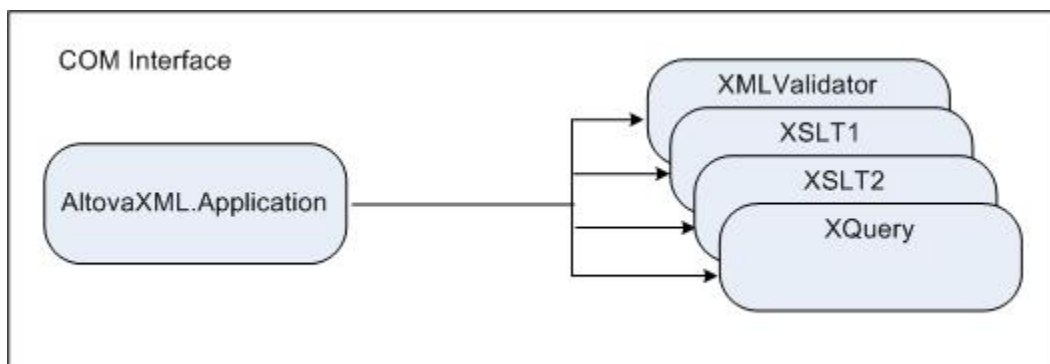
If the `AltovaXML_COM.exe` has been manually registered and you now wish to unregister it, then it should be manually unregistered. To manually unregister AltovaXML, call the application with the `/unregserver` parameter. For example, if the AltovaXML executable is in the folder `c:\AltovaXML`, then open a Windows Command Prompt window, key in `c:\AltovaXML\AltovaXML_COM.exe /unregserver`, and press **Enter**. You can check the Registry Editor for confirmation of unregistration.

**Note:** If AltovaXML was registered by the installer, the unregistration should be done by the installer—that is, by de-installing AltovaXML from the machine.

## 2.2.2 AltovaXML Object Model

The starting point for using the functionality of AltovaXML is the Application interface. This object contains the four objects that provide the AltovaXML functionality: XML validation, XSLT 1.0 transformations, XSLT 2.0 transformations, and XQuery 1.0 document processing. These objects have dual interfaces: the Dispatch Interface and the Raw Interface, which enables them to be used in scripting languages as well as in applications.

The object model of the AltovaXML API is depicted in the following diagram.



The hierarchy of the object model is shown below, and the five interfaces are described in detail in the corresponding sections. The properties and usage of each interface are described in the section for that interface.

- [Application](#)
  - [XMLValidator](#)
  - [XSLT1](#)
  - [XSLT2](#)
  - [XQuery](#)

### Note:

Note the following general points about COM Interface usage:

- The term XML document refers not only to an XML document contained in an XML file but also to an XML document created with the `InputXMLFromText` property.
- Properties that take a resource location as its input accept absolute paths, as well as the HTTP and FTP protocols.
- When relative paths are used by a method to locate a resource, the resolution of the relative path should be defined in the calling module.

## 2.2.3 Application

### Description

AltovaXML. Application is the root for all other objects. It is the only object you can create with the `CreateObject` function (of VisualBasic) or other similar COM-related functions.

### Properties

AltovaXML. Application has the four properties listed below. Each of these functions returns the interface for the specific component. The details of each interface are given in the respective sections listed below.

- [XMLValidator](#)
- [XSLT1](#)
- [XSLT2](#)
- [XQuery](#)

### Methods

The following methods, which are available on the application object, enable the addition of catalogs used for document lookup. After catalogs are added they are used for lookup till the COM server terminates. Added catalogs cannot be removed.

```
app.AddXMLCatalogDefault()
```

Adds Altova's default `RootCatalog.xml` to the catalogs

```
app.AddXMLCatalogFromFile( string catalogfilename )
```

Adds the catalog identified by `catalogfilename` to the catalogs

```
app.AddXMLCatalogFromText( string catalogtext )
```

Adds the catalog with content `catalogtext` to the catalogs

### Examples

Given below is a Visual Basic script that first creates the AltovaXML object, and then calls properties of the application interface.

```
Sub CommandButton1_Click()  
Set objAltovaXML = CreateObject("AltovaXML.Application")  
  
    objAltovaXML.XMLValidator.InputXMLFileName =  
"c:\AltovaXML\test.xml"  
    Sheet1.Cells(5, 2) = objAltovaXML.XMLValidator.IsValid  
  
    objAltovaXML.XSLT1.InputXMLFromText = "<?xml version='1.0'  
encoding='UTF-8' ?><a><b/></a>"  
    objAltovaXML.XSLT1.XSLFileName = "c:\workarea\altova_xml\1.xslt"  
    Sheet1.Cells(6, 2) =  
objAltovaXML.XSLT1.ExecuteAndGetResultAsString  
  
End Sub
```

## 2.2.4 XMLValidator

### Description

The `XMLValidator` interface provides methods to test:

- The well-formedness of an XML document.
- The validity of an XML document against a DTD or XML Schema referenced from within the XML document.
- The validity of an XML document against a DTD or XML Schema supplied externally via the code.
- The validity of an XBRL document against an XBRL taxonomy (a `.xsd` file).

All these methods return Boolean `TRUE` or `FALSE`. See *examples below*.

**Note:** Where string inputs are to be interpreted as URLs, absolute paths should be used. If a relative path is used, a mechanism to resolve the relative path should be defined in the calling module.

### Methods

The following methods are available:

**IsWellFormed()** as Boolean

`IsWellFormed` checks the well-formedness of the XML document. Returns `TRUE` if the XML document is well-formed, `FALSE` if it is not well-formed.

**IsValid()** as Boolean

`IsValid` validates the XML document against the DTD or XML Schema referenced in the XML document. Returns `TRUE` if the XML document is valid, `FALSE` if invalid. To validate against a DTD or XML Schema not referenced in the XML document, use the method `IsValidWithExternalSchemaOrDTD`.

**IsValidWithExternalSchemaOrDTD()** as Boolean

`IsValidWithExternalSchemaOrDTD` validates the XML document against the DTD or XML Schema supplied by any one of the following properties: `SchemaFileName`, `DTDFileName`, `SchemaFromText`, or `DTDFromText`. If more than one of these properties has values set for it, then the `IsValidWithExternalSchemaOrDTD` method uses the property that has been set last. Returns `TRUE` if the XML document is valid, `FALSE` if invalid. To validate against a DTD or XML Schema referenced in the XML document, use the method `IsValid`.

**Note:** Validation and well-formedness checks must always occur after assigning the XML and/or DTD or XML Schema document to the respective properties.

### Properties

The following properties are defined:

**InputXMLFileName**

A string input that is read as a URL to locate the XML file to be validated.

**SchemaFileName**

A string input that is read as a URL to locate the XML Schema file against which the XML document is to be validated.

**DTDFileName**

A string input that is read as a URL to locate the DTD file against which the XML document is to be validated.



**InputXMLFromText**

A string input that constructs an XML document.

**SchemaFromText**

A string input that constructs an XML Schema document.

**DTDFromText**

A string input that constructs a DTD document.

**LastErrorMessage**

Returns the last error message.

**TreatXBRLInconsistenciesAsErrors**

If set to `True`, returns XBRL semantic inconsistencies as errors. Default is `False`.

**Examples**

Given below is a single Visual Basic procedure that shows how the methods and properties of the `XMLValidator` interface can be used. This code is intended for use as a macro in an MS Excel worksheet, and references to worksheet cells indicate locations of input or output data. The file `c:\AltovaXML\test.xml` is assumed to contain a reference to a DTD.

```
Sub CommandButton1_Click()  
Set objAltovaXML = CreateObject("AltovaXML.Application")  
  
    objAltovaXML.XMLValidator.InputXMLFromText = "<?xml version='1.0'  
encoding='UTF-8' ?><a><b/></a>"  
    Sheet1.Cells(4, 2) = objAltovaXML.XMLValidator.IsWellFormed  
  
    objAltovaXML.XMLValidator.InputXMLFileName = "c:\AltovaXML\test.xml"  
    Sheet1.Cells(5, 2) = objAltovaXML.XMLValidator.IsValid  
  
    objAltovaXML.XMLValidator.InputXMLFileName = "c:\AltovaXML\test.xml"  
    objAltovaXML.XMLValidator.DTDFileName = "c:\AltovaXML\test.dtd"  
    Sheet1.Cells(6, 2) =  
objAltovaXML.XMLValidator.IsValidWithExternalSchemaOrDTD  
  
    objAltovaXML.XMLValidator.InputXMLFromText = "<?xml version='1.0'  
encoding='UTF-8' ?><a><b/></a>"  
    objAltovaXML.XMLValidator.DTDFileName = "c:\AltovaXML\test.dtd"  
    Sheet1.Cells(7, 2) =  
objAltovaXML.XMLValidator.IsValidWithExternalSchemaOrDTD  
End Sub
```

## 2.2.5 XSLT1

### Description

The `XSLT1` interface provides methods and properties to execute an XSLT 1.0 transformation using the Altova XSLT 1.0 Engine. Results can be saved to a file or returned as a string. The interface also enables XSLT parameters to be passed to the XSLT stylesheet. The URLs of XML and XSLT files can be supplied as strings via interface properties. Alternatively, the XML and XSLT documents can be constructed within the scripting or programming code as text strings. See *examples below*.

**Note:** Where string inputs are to be interpreted as URLs, absolute paths should be used. If a relative path is used, a mechanism to resolve the relative path should be defined in the calling module.

### Methods

The following methods are available:

**Execute**(OutputFileName as String)

void **execute**(String outputFilename)

**Execute** executes an XSLT 1.0 transformation and saves the result to an output file, the name and location of which is provided as an input string to the **Execute** method. For example:

```
Execute("C:\OutputDoc.xml") .
```

**ExecuteAndGetResultAsString**( ) as String

**ExecuteAndGetResultAsString** executes an XSLT 1.0 transformation and returns the result as a UTF-16 text string.

**AddExternalParameter**(ParamName as String, ParamValue as String)

Takes a parameter name and the value of this parameter as input arguments. Each external parameter and its value is to be specified in a separate call to the method. If multiple calls specify the same parameter name the value set by the latest will be used. Since parameter values are XPath expressions, parameter values that are strings must be enclosed in single quotes. In this example, two parameter values are submitted:

```
AddExternalParameter("Param1", "' http://www.altova.com/'");  
AddExternalParameter("Param2", "concat(' http://www.altova.com/',  
MyFile/@url)");
```

*Also see examples below.*

**ClearExternalParameterList**( )

No argument should be provided. The **ClearExternalParameterList** clears the external parameters list created with **AddExternalParameter** methods.

**Note:** Transformation must always occur after assigning the XML and XSLT documents.

### Properties

The following properties are defined:

**InputXMLFileName**

A string input that is read as a URL to locate the XML file to be transformed.

**XSLFileName**

A string input that is read as a URL to locate the XSLT file to be used for the transformation.

**InputXMLFromText**

A string input that constructs an XML document.

**XSLFromText**

A string input that constructs an XSLT document.

**XSLStackSize**

The stack size is the maximum depth of executed instructions. The stack size can be changed with the `XSLStackSize` property. The minimum allowed stack size is 100. The default stack size is 1000. If the stack size is exceeded during a transformation, an error is reported.

**LastErrorMessage**

Returns the last error message.

**JavaExtensionsEnabled**

Enables Java extensions. You can specify whether Java extensions should be enabled or not by submitting `true` or `false` (case-insensitive) as a `VARIANT_BOOL` argument.

**DotNetExtensionsEnabled**

Enables .NET extensions. You can specify whether .NET extensions should be enabled or not by submitting `true` or `false` (case-insensitive) as a `VARIANT_BOOL` argument.

**Examples**

Given below is a single Visual Basic procedure that shows how the various methods and properties of the `XSLT1` interface can be used. This code is intended for use as a macro in an MS Excel worksheet, and references to worksheet cells indicate locations of input or output data.

```
Sub CommandButton1_Click()  
Set objAltovaXML = CreateObject("AltovaXML.Application")  
  
objAltovaXML.XSLT1.InputXMLFromText = "<?xml version='1.0'  
encoding='UTF-8' ?>  
<a><b/></a>"  
objAltovaXML.XSLT1.XSLFileName = "c:\AltovaXML\test.xslt"  
objAltovaXML.XSLT1.Execute "c:\AltovaXML\test_result.xml"  
  
objAltovaXML.XSLT1.XSLStackSize = "500"  
objAltovaXML.XSLT1.InputXMLFromText = "<?xml version='1.0'  
encoding='UTF-8' ?>  
<company><name/><year>2005</year></company>"  
objAltovaXML.XSLT1.XSLFileName = "c:\AltovaXML\test.xslt"  
objAltovaXML.XSLT1.AddExternalParameter "web", "' www.altova.com' "  
objAltovaXML.XSLT1.AddExternalParameter "year", "/company/year"  
Sheet1.Cells(6, 2) = objAltovaXML.XSLT1.ExecuteAndGetResultAsString  
objAltovaXML.XSLT1.ClearExternalParameterList  
objAltovaXML.XSLT1.AddExternalParameter "web", "' www.nanonull.com' "  
objAltovaXML.XSLT1.AddExternalParameter "year", "/company/year"  
Sheet1.Cells(7, 2) = objAltovaXML.XSLT1.ExecuteAndGetResultAsString  
End Sub
```

## 2.2.6 XSLT2

### Description

The `XSLT2` interface provides methods and properties to execute an XSLT 2.0 transformation using the Altova XSLT 2.0 Engine. Results can be saved to a file or returned as a string. The interface also enables XSLT parameters to be passed to the XSLT stylesheet. The URLs of XML and XSLT files can be supplied as strings via interface properties. Alternatively, the XML and XSLT documents can be constructed within the scripting or programming code as text strings. See *examples below*.

### Note:

- Where string inputs are to be interpreted as URLs, absolute paths should be used. If a relative path is used, a mechanism to resolve the relative path should be defined in the calling module.
- The XSLT 2.0 Engine can be used in its backward compatibility mode to process an XSLT 1.0 stylesheet. The output, however, could be different than that produced by the XSLT 1.0 Engine processing the same XSLT 1.0 stylesheet.

### Methods

The following methods are available:

#### **Execute**(OutputFileName as String)

void **execute**(String outputFilename)

`Execute` executes an XSLT 2.0 transformation and saves the result to an output file, the name and location of which is provided as an input string to the `Execute` method. For example:

```
Execute("C:\OutputDoc.xml").
```

#### **ExecuteAndGetResultAsString**( ) as String

`ExecuteAndGetResultAsString` executes an XSLT 2.0 transformation and returns the result as a UTF-16 text string.

#### **AddExternalParameter**(ParamName as String, ParamValue as String)

Takes a parameter name and the value of this parameter as input arguments. Each external parameter and its value is to be specified in a separate call to the method. If multiple calls specify the same parameter name the value set by the latest will be used. Since parameter values are XPath expressions, parameter values that are strings must be enclosed in single quotes. See *examples below*. Notice in the examples that the `date` parameter is given a value that is an XPath 2.0 function (`current-date()`). In this example, two parameter values are submitted:

```
AddExternalParameter("Param1", "' http://www.altova.com/'");  
AddExternalParameter("Param2", "concat(' http://www.altova.com/',  
MyFile/@url)");
```

Also see *examples below*.

#### **ClearExternalParameterList**( )

No argument should be provided. The `ClearExternalParameterList` clears the external parameters list created with `AddExternalParameter` methods.

#### **InitialTemplateName**

Sets the initial named template. The argument is the name of the template from which processing is to start. For example: `InitialNamedTemplate = "MyNamedTemplate"`.

#### **InitialTemplateMode**

Sets the initial mode for processing. The argument is the name of the required initial mode.

Templates with this mode value will be processed. For example:

```
InitialTemplateMode="MyMode".
```

**Note:** Transformation must always occur after assigning the XML and XSLT documents.

### Properties

The following properties are defined:

#### **InputXMLFileName**

A string input that is read as a URL to locate the XML file to be transformed.

#### **XSLFileName**

A string input that is read as a URL to locate the XSLT file to be used for the transformation.

#### **InputXMLFromText**

A string input that constructs an XML document.

#### **XSLFromText**

A string input that constructs an XSLT document.

#### **XSLStackSize**

The stack size is the maximum depth of executed instructions. The stack size can be changed with the `XSLStackSize` property. The minimum allowed stack size is 100. The default stack size is 1000. If the stack size is exceeded during a transformation, an error is reported.

#### **LastErrorMessage**

Returns the last error message.

#### **JavaExtensionsEnabled**

Enables Java extensions. You can specify whether Java extensions should be enabled or not by submitting `true` or `false` (case-insensitive) as a `Variant_Bool` argument.

#### **DotNetExtensionsEnabled**

Enables .NET extensions. You can specify whether .NET extensions should be enabled or not by submitting `true` or `false` (case-insensitive) as a `Variant_Bool` argument.

### Examples

Given below is a single Visual Basic procedure that shows how the various methods and properties of the `XSLT2` interface can be used. This code was intended for use as a macro in an MS Excel worksheet, and references to worksheet cells indicate locations of input or output data.

```
Sub CommandButton1_Click()  
Set objAltovaXML = CreateObject("AltovaXML.Application")  
  
objAltovaXML.XSLT2.InputXMLFromText = "<?xml version='1.0'  
encoding=' UTF-8' ?>  
<a><b/></a>"  
objAltovaXML.XSLT2.XSLFileName = "c:\AltovaXML\test.xslt"  
Sheet1.Cells(7, 2) = objAltovaXML.XSLT2.ExecuteAndGetResultAsString  
  
objAltovaXML.XSLT2.XSLStackSize = "500"  
objAltovaXML.XSLT2.InputXMLFromText = "<?xml version='1.0'  
encoding=' UTF-8' ?>  
<company><name/><year>2005</year></company>"  
objAltovaXML.XSLT2.XSLFileName = "c:\workarea\AltovaXML\2.xslt"
```

```
objAltovaXML.XSLT2.AddExternalParameter "date", "current-date()"
objAltovaXML.XSLT2.AddExternalParameter "hq", "' Vienna, Austria'"
Sheet1.Cells(8, 2) = objAltovaXML.XSLT2.ExecuteAndGetResultAsString
objAltovaXML.XSLT2.AddExternalParameter "web", "' www.nanonull.com'"
objAltovaXML.XSLT2.AddExternalParameter "year", "/company/year"
objAltovaXML.XSLT2.Execute "c:\workarea\AltovaXML\test_result_xslt2.xml"
Sheet1.Cells(9, 2) = objAltovaXML.XSLT2.ExecuteAndGetResultAsString
End Sub
```

## 2.2.7 XQuery

### Description

The `XQuery` interface provides methods and properties to execute an XQuery 1.0 transformation using the Altova XQuery 1.0 Engine. Results can be saved to a file or returned as a string. The interface also enables external XQuery variables to be passed to the XQuery document. The URLs of XQuery and XML files can be supplied as strings via interface properties. Alternatively, the XML and XQuery documents can be constructed within the scripting or programming code as text strings. *See examples below.*

**Note:** Where string inputs are to be interpreted as URLs, absolute paths should be used. If a relative path is used, a mechanism to resolve the relative path should be defined in the calling module.

### Methods

The following methods are available:

**Execute**(OutputFileName as String)

void **execute**(String outputFilename)

`Execute` executes an XQuery 1.0 transformation and saves the result to an output file, the name and location of which is provided as an input string to the `Execute` method. For example: `Execute("C:\OutputDoc.xml")`.

**ExecuteAndGetResultAsString**( ) as String

`ExecuteAndGetResultAsString` executes an XQuery 1.0 transformation and returns the result as a UTF-16 text string.

**AddExternalVariable**(VarName as String, VarValue as String)

Takes a variable name and the value of this variable as input arguments. Each external variable and its value is to be specified in a separate call to the method. Variables must be declared in the XQuery document, optionally with a type declaration. Whatever the type declaration for the external variable in the XQuery document, the variable value submitted to the `AddExternalVariable` method does not need any special delimiter, such as quotes (*see example below*). However, the lexical form must match that of the expected type (for example, a variable of type `xs:date` must have a value in the lexical form `2004-01-31`; a value in the lexical form `2004/Jan/01` will cause an error). Note that this also means that you cannot use an XQuery 1.0 function (for example, `current-date()`) as the value of an external variable (since the lexical form of the function as it is written will either not match the required data type (if the datatype is specified in the declaration of the external variable) or will be read as a string (if the datatype is not specified).) If multiple calls specify the same variable name the value set by the latest will be used.

**AddExternalVariableAsXPath**(VarName as String, VarValue as String)

Takes a variable name and the value of this variable as input arguments. Similar to `AddExternalVariable` method, except that `AddExternalVariableAsXPath` will be evaluated as an XPath 2.0 expression. This makes it possible to pass in nodes and sequences with more than one element.

**ClearExternalVariableList**

No argument should be provided. The `ClearExternalVariableList` clears the external variables list created with `AddExternalVariable` methods.

**Note:** Setting the optional XML document must always be done before query execution.

### Properties

*The following properties are defined:*

**XQueryFileName**

A string input that is read as a URL to locate the XQuery file to be executed. If both the `XQueryFileName` property and `XQueryFromText` property are specified, then the property that has been set later than the other (in the code sequence) is used.

**InputXMLFileName**

A string input that is read as a URL to locate the XML file that will be loaded into the query. XQuery navigation expressions are evaluated with reference to the document node of this XML document. If both the `InputXMLFileName` property and `InputXMLFromText` property are specified, then the property that has been set later than the other (in the code sequence) is used.

**XQueryFromText**

A string input that constructs an XQuery document. If both the `XQueryFileName` property and `XQueryFromText` property are specified, then the property that has been set later than the other (in the code sequence) is used.

**InputXMLFromText**

A string input that constructs an XML document. XQuery navigation expressions are evaluated with reference to the document node of this XML document. If both the `InputXMLFileName` property and `InputXMLFromText` property are specified, then the property that has been set later than the other (in the code sequence) is used.

**LastErrorMessage**

Returns the last error message.

**JavaExtensionsEnabled**

Enables Java extensions. You can specify whether Java extensions should be enabled or not by submitting `true` or `false` (case-insensitive) as a `Variant_Bool` argument.

**DotNetExtensionsEnabled**

Enables .NET extensions. You can specify whether .NET extensions should be enabled or not by submitting `true` or `false` (case-insensitive) as a `Variant_Bool` argument.

**Note:** If an XML document is set and is not needed for a new XQuery execution, then it should be cleared with an empty string assignment.

*The following serialization options are defined:*

**OutputMethod**

The required output method can be specified by submitting the required value as a string argument. Valid values are: `xml`, `xhtml`, `html`, and `text`. For example:

```
objAltovaXML.XQuery.OutputMethod = "xml".
```

If the value is invalid, it is ignored. The default output method is `xml`.

**OutputOmitXMLDeclaration**

You can specify whether the XML declaration should be omitted or included in the output by submitting `true` or `false` (case-insensitive) as a Boolean argument. For example:

```
objAltovaXML.XQuery.OutputOmitXMLDeclaration = "FALSE".
```

If the value is invalid, an error is raised. The default option is `TRUE`.

**OutputIndent**

You can specify whether the output should be indented or not by submitting `true` or `false` (case-insensitive) as a Boolean argument. For example:



`objAltovaXML.XQuery.OutputIndent = "TRUE"`. If the value is invalid, an error is raised. The default option is `False`.

#### OutputEncoding

The required output encoding can be specified by submitting the encoding value as a string argument. For example: `objAltovaXML.XQuery.OutputEncoding = "UTF-8"`. If the value is invalid, it is ignored. The default output encoding is UTF-8.

**Note:** For the serialization options, Raw Interface and Dispatch Interface usage differs. In the Raw Interface, if no argument is provided with these properties, then the current value of the property is returned. You would use something like: `put_OutputOption( VARIANT_BOOL bVal )` or `VARIANT_BOOL bVal = get_OutputOption()`, respectively, to set values and get values. In the Dispatch Interface, you can use `b = myXQuery.OutputOption` to get values and `myXQuery.OutputOption = b` to set values. For example, in the Dispatch Interface, `Sheet1.Cells(10, 2) = objAltovaXML.XQuery.OutputEncoding` would get the current output encoding.

#### Examples

Given below is a single Visual Basic procedure that shows how the various methods and properties of the `XQuery` interface can be used. This code was intended for use as a macro in an MS Excel worksheet, and references to worksheet cells indicate locations of input or output data.

```
Sub CommandButton1_Click()
Set objAltovaXML = CreateObject("AltovaXML.Application")

objAltovaXML.XQuery.InputXMLFileName = "c:\AltovaXML\test.xml"
objAltovaXML.XQuery.XQueryFromText = " xquery version '1.0';
  declare variable $string as xs:string external;
  declare variable $num as xs:decimal external;
  declare variable $date as xs:date external;
  $string, ' ', 2*$num, ' ', $date "
objAltovaXML.XQuery.AddExternalVariable "string", "A string"
objAltovaXML.XQuery.AddExternalVariable "num", "2.1"
objAltovaXML.XQuery.AddExternalVariable "date", "2005-04-21"
Sheet1.Cells(10, 2) = objAltovaXML.XQuery.OutputEncoding
objAltovaXML.XQuery.OutputMethod = "text"
Sheet1.Cells(11, 2) = objAltovaXML.XQuery.OutputMethod
objAltovaXML.XQuery.OutputIndent = "TRUE"
Sheet1.Cells(12, 2) = objAltovaXML.XQuery.OutputIndent
objAltovaXML.XQuery.OutputOmitXMLDeclaration = "FALSE"
Sheet1.Cells(13, 2) = objAltovaXML.XQuery.OutputOmitXMLDeclaration
Sheet1.Cells(14, 2) = objAltovaXML.XQuery.ExecuteAndGetResultAsString
End Sub
```

## 2.2.8 Examples

This section contains example code in (i) Visual Basic for an Excel macro; (ii) JScript; and (iii) C++. These examples will give you an idea of how you can use AltovaXML with a COM Interface.

For more detailed examples, see the example files in the `Examples` folder in the application folder.

### Visual Basic

The following Visual Basic example is the code for a macro in an Excel worksheet (*screenshot below*). The macro has been assigned to the button `Run Expressions`. On clicking the button, the Visual Basic code is executed.

	A	B
1	<b>XQuery or XML in Application</b>	<b>Result</b>
2	element a {for \$i in (-3 to 3) return -\$i}	<a>3 2 1 0 -1 -2 -3</a>
3	<node>6;154;738-34</node>	6.154.738 34
4		A code-generated string
5	Run Expressions	

### Code sample

The Visual Basic code below uses the `XQuery` interface.

```
Sub CommandButton1_Click()
Set objAltovaXML = CreateObject("AltovaXML.Application")

objAltovaXML.XQuery.XQueryFromText = Sheet1.Cells(2, 1)
Sheet1.Cells(2, 2) = objAltovaXML.XQuery.ExecuteAndGetResultAsString

objAltovaXML.XQuery.InputXMLFromText = Sheet1.Cells(3, 1)
objAltovaXML.XQuery.XQueryFromText = "translate(node, ';'-'', '.' ')"
Sheet1.Cells(3, 2) = objAltovaXML.XQuery.ExecuteAndGetResultAsString

objAltovaXML.XQuery.InputXMLFromText = "<a myAttr=' A code-generated
string' />"
objAltovaXML.XQuery.XQueryFromText = "string(/a/@*)"
Sheet1.Cells(4, 2) = objAltovaXML.XQuery.ExecuteAndGetResultAsString
End Sub
```

On clicking the button **Run Expressions** in the Excel worksheet, the following three XQuery instructions are executed:

1. The input for the `XQueryFromText` property is an XQuery expression taken as text from the Excel worksheet cell 2A. The `ExecuteAndGetResultAsString` property executes the XQuery expression and places the result in the Excel worksheet cell 2B.
2. The input for the `InputXMLFromText` property is an XML fragment taken from the Excel worksheet cell 3A. The XQuery expression is given to the `XQueryFromText` property directly in the code. The result is placed in the Excel worksheet cell 3B.
3. The `InputXMLFromText` property creates an XML tree from the XML fragment provided to it. The XQuery expression is given to the `XQueryFromText` property directly in the code, and the result is placed in the Excel worksheet cell 4B.

## JScript

Given below is a JScript code sample that shows how AltovaXML can be used via the COM interface.

### Code sample

```
// ////////////////////////////////// global variables //////////////////////////////////
var objAltovaXML = null;

// ////////////////////////////////// Helpers //////////////////////////////////

function Exit(strErrorText)
{
    WScript.Echo(strErrorText);

    if (objAltovaXML != null)
        objAltovaXML.Quit();

    WScript.Quit(-1);
}

function ERROR(strText, objErr)
{
    if (objErr != null)
        Exit ("ERROR: (" + (objErr.number & 0xffff) + ") " + objErr.description +
" - " + strText);
    else
        Exit ("ERROR: " + strText);
}

function CreateGlobalObjects ()
{
    // create the AltovaXML connection
    // if there is a running instance of AltovaXML (that never had a
connection) - use it
    // otherwise, we automatically create a new instance
    try
    {
        objAltovaXML = WScript.GetObject("", "AltovaXML.Application");
        //WScript.Echo("Successfully accessing AltovaXML.Application");
    }
    catch(err)
    {
        WScript.Echo(err)
        { Exit("Can't access or create AltovaXML.Application"); }
    }
}

// ////////////////////////////////// MAIN //////////////////////////////////

CreateGlobalObjects();

objAltovaXML.XQuery.InputXMLFromText = " \
<bib> \
<book year=\"1994\"> \
    <title>TCP/IP Illustrated</title> \
    <author><last>Stevens</last><first>W.</first></author> \
    <publisher>AW</publisher> \
    <price>65.95</price> \
</book> \
<book year=\"1992\"> \
    <title>Advanced Programming in the Unix Environment</title> \
    <author><last>Stevens</last><first>W.</first></author> \
```

```

    <publisher>AW</publisher> \
    <price>65.95</price> \
  </book> \
  <book year="2000"> \
    <title>Data on the Web</title> \
    <author><last>Abiteboul</last><first>Serge</first></author> \
    <author><last>Abiteboul</last><first>Serge</first></author> \
    <author><last>Abiteboul</last><first>Serge</first></author> \
    <publisher>John Jameson Publishers</publisher> \
    <price>39.95</price> \
  </book> \
  <book year="1999"> \
    <title>Digital TV</title> \

<editor><last>Gassy</last><first>Viktor</first><affiliation>CITI</affiliation>
</editor> \
  <publisher>Kingston Academic Press</publisher> \
  <price>129.95</price> \
</book> \
</bib> ";

```

```

objAltovaXML.XQuery.XQueryFromText = "\
(: Filename: xmpQ1.xq :) \
(: Source: http://www.w3.org/TR/xquery-use-cases/#xmp-data :) \
(: Section: 1.1.1.9 Q1 :) \
(: List books published by AW after 1991, including their year and title.:)
\
<bib> \
{ \
  for $b in /bib/book where $b/publisher = "AW" and $b/@year > 1991 \
    return <book year="{ $b/@year }" { $b/title } </book>
\
} \
</bib> ";

```

```

var sResult = objAltovaXML.XQuery.ExecuteAndGetResultAsString();
WScript.Echo(sResult);

```

## C++

Given below is a C++ code sample that shows how AltovaXML can be used via the COM interface.

### Code sample

```

// TestAltovaXML.cpp : Defines the entry point for the console application.
//
#include "objbase.h"
#include <iostream>
#include "atlbase.h"

#import "AltovaXML_COM.exe" no_namespace raw_interfaces_only
// - or -
// #import "AltovaXML_COM.exe" raw_interfaces_only
// using namespace AltovaXMLLib;

int main(int argc, char* argv[])
{
    HRESULT hr = S_OK;
    hr = CoInitialize(NULL);
    if ( hr == S_OK )
    {
        IApplicationPtr ipApplication;

```

```
        hr = CoCreateInstance(
            __uuidof( Application
        ),
            NULL,
            CLSCTX_ALL,
            __uuidof( IApplication),

reinterpret_cast<void**>( &ipApplication)
    );
    if ( hr == S_OK )
    {
        IXQueryPtr ipXQuery;
        hr = ipApplication->get_XQuery( &ipXQuery );

        if ( hr == S_OK )
        {
            CComBSTR sXQExpr( "(1 to 10)[. mod 2 != 0]" );
            BSTR bstrResult;

            hr = ipXQuery->put_XQueryFromText( sXQExpr );
            hr = ipXQuery->ExecuteAndGetResultAsString(
&bstrResult );

            std::cout << ( char* )_bstr_t( bstrResult ) <<
std::endl;

            ipXQuery. Release();
        }
        ipApplication. Release();
    }
    CoUninitialize();
}
return 0;
}
```

## 2.3 Java Interface

The AltovaXML API can be accessed from Java code. To allow accessing the AltovaXML automation server directly from Java code, the libraries listed below must reside in the classpath. They are installed in the folder: `JavaAPI` in the AltovaXML application folder.

- `AltovaAutomation.dll`: a JNI wrapper for Altova automation servers
- `AltovaAutomation.jar`: Java classes to access Altova automation servers
- `AltovaXMLAPI.jar`: Java classes that wrap the AltovaXML automation interface
- `AltovaXMLAPI_JavaDoc.zip`: a Javadoc file containing help documentation for the Java API

**Note:** In order to use the Java API, the DLL and Jar files must be on the Java Classpath, and `AltovaXML_COM.exe` must be [registered as a COM server object](#).

### Example Java project

An example Java project is supplied with your product installation. You can test the Java project and modify and use it as you like. Note, however, that the folder might be write-protected, in which case you must change your user-access settings. For more details of the example Java project, see the section, [Example Java Project](#).

### Previous interface

The previous Java interface for AltovaXML has been replaced since Version 2012. Documentation for the old interface can be found here: [Old Java API \(Obsolete\)](#).

### Differences between old and new interfaces

The differences between the old (pre-v2012) interface and the new (v2012 onwards) interface are listed below:

- The new `AltovaXMLFactory()` is used to create an instance of [AltovaXMLFactory](#) instead of [AltovaXMLFactory.getInstance\(\)](#)
- The methods of [AltovaXMLFactory](#) to access the XML Validator, XQuery, XSLT1, and XSLT2 engines are called `getXMLValidator()`, `getXQuery()`, `getXSLT1()`, `getXSLT2()`
- To free an engine instance or the factory use `dispose()` instead of `releaseInstance()`
- The method `enableJavaExtension` is now called `setJavaExtensioEnabled` in the various engine interfaces
- The method `enableDotNetExtension` is now called `setDotNetExtensioEnabled` in the various engine interfaces

### 2.3.1 Example Java Project

The AltovaXML installation package contains an example Java project, located in the Java folder of the API Examples folder: `AltovaXMLExamples/API/Java` in the AltovaXML application folder.

This folder contains Java examples for the AltovaXML API. You can test it directly from the command line using the batch file `BuildAndRun.bat` (located in the same folder) or you can compile and run the example project from within Eclipse. See below for instructions on how to use these procedures.

#### File list

The Java examples folder contains all the files required to run the example project. These files are listed below:

<code>AltovaAutomation.dll</code>	Java-COM bridge: DLL part
<code>AltovaAutomation.jar</code>	Java-COM bridge: Java library part
<code>AltovaXMLAPI.jar</code>	Java classes of the AltovaXML API
<code>UseAltovaXML.java</code>	Java example source code
<code>BuildAndRun.bat</code>	Batch file to compile and run example code from the command line prompt. Expects folder where Java Virtual Machine resides as parameter.
<code>.classpath</code>	Eclipse project helper file
<code>.project</code>	Eclipse project file
<code>AltovaXMLAPI_JavaDoc.zip</code>	Javadoc file containing help documentation for the Java API

#### Running the example from the command line

To run the example from the command line, open a command prompt window, go to the Java folder of the API Examples folder (*see above for location*), and then type:

```
buildAndRun.bat "<Path-to-the-Java-bin-folder>"
```

The Java binary folder must be that of a JDK 1.5 or later installation on your computer.

Press the **Return** key. The Java source in `UseAltovaXML.java` will be compiled and then executed.

#### Loading the example in Eclipse

Open Eclipse and use the **Import | Existing Projects into Workspace** command to add the Eclipse project file (`.project`) located in the Java folder of the API Examples folder (*see above for location*). The project `UseAltovaXML` will then appear in your Package Explorer or Navigator.

Select the project and then the command **Run as | Java Application** to execute the example.

**Note:** You can select a class name or method of the Java API and press F1 to get help for that class or method.

**Java source code listing**

The Java source code in the example file `UseAltovaXML.java` is listed below with comments.

```

01 import com.altova.automation.AltovaXML.*;
02 import com.altova.automation.libs.AutomationException;
03
04
05 public class UseAltovaXML
06 {
07
08     /**
09      * @param args
10      */
11     public static void main(String[] args)
12     {
13         // Locate samples installed with the product.
14         // REMARK: You will need to modify this if you use a different major version, use
the 64-bit
15         //          version of the program, or installed AltovaXML in a different
location.
16         String strExamplesFolder = System.getenv("ProgramFiles") +
"/Altova/AltovaXML2012/AltovaXMLExamples/";
17
18         String inFilename = strExamplesFolder + "simple.xml";
19         String xqFilename = strExamplesFolder + "CopyInput.xq";
20         System.out.println("AltovaXML Java JNI XQuery");
21
22         AltovaXMLFactory xmlFactory = null;
23         try
24         {
25             // Get application instance
26             xmlFactory = new AltovaXMLFactory();
27
28             // Get XML validator and XQ method pointers from the application instance
29             XMLValidator validator = xmlFactory.getXMLValidator();
30             XQuery xQuery = xmlFactory.getXQuery();
31
32             // We only want to work with input files that are well-formed.
33             validator.setInputXMLFileName(inFilename);
34             if ( validator.isWellFormed() )
35             {
36                 // If the file is well-formed, copy it using XQuery
37                 xQuery.setInputXMLFileName(inFilename);
38                 xQuery.setXQueryFileName( xqFilename );
39
40                 // Test return value
41                 String resultString = xQuery.executeAndGetResultAsString();
42                 if ( resultString == null )
43                     System.out.println("XQuery error: " + xQuery.getLastErrorMessage());
44                 else
45                     System.out.println("Transform contents: " + resultString );
46             }
47             else
48                 System.out.println("Not wellformed error: " + validator.getLastErrorMessage(
));
49         }
50     }
51     catch ( AutomationException e )
52     {
53         // An error occurred when talking to the AltovaXML COM interface.
54         System.out.println("Error accessing AltovaXML: " + e.getMessage());
55     }
56     finally
57     {
58         // Now we can release the factory to immediately shut-down AltovaXML_COM.exe
59         if ( xmlFactory != null )
60             xmlFactory.dispose();
61     }
62 }
63 }

```





## 2.3.2 Classes

Given below is a summary of the classes of `com.altova.engines`. Detailed descriptions are given in the respective sections.

- [AltovaXMLFactory](#)  
Creates new AltovaXML COM server object instance via native call, and provides access to AltovaXML engines.
- [XMLValidator](#)  
Class holding XMLValidator.
- [XQuery](#)  
Class holding the XQuery 1.0 Engine.
- [XSLT1](#)  
Class holding the XSLT 1.0 Engine.
- [XSLT2](#)  
Class holding the XSLT 2.0 Engine.

### AltovaXMLFactory

```
public class AltovaXMLFactory
extends java.lang.Object
```

#### Description

Use `AltovaXMLFactory()` to create a new AltovaXML COM server object instance. This provides access to the AltovaXML engines. The relationship between `AltovaXMLFactory` and the AltovaXML COM object is one-to-one. This means that subsequent calls to the `getENGINEName()` function will return interfaces for the same engine instance.

#### Methods

The following methods are defined. The methods for catalogs enable the addition of catalogs used for document lookup. After catalogs are added they are used for lookup till the COM server terminates. Added catalogs cannot be removed.

```
addXMLCatalogDefault()
public void addXMLCatalogDefault()
Adds Altova's default RootCatalog.xml to the catalogs.
```

```
addXMLCatalogFromFile( string catalogfilename )
public void addXMLCatalogFromFile(java.lang.String
bstrXMLCatalogFileName)
Adds the catalog identified by catalogfilename to the catalogs.
```

```
addXMLCatalogFromText( string catalogtext )
public void addXMLCatalogFromText(java.lang.String bstrXMLCatalogText)
Adds the catalog with content catalogtext to the catalogs.
```

```
dispose
public void dispose()
Releases the object's connection to the COM server.
```

**getXMLValidator**

```
public XMLValidator getXMLValidator()
```

Retrieves the XMLValidator. The object's connection to the engine must be released after use. To do this, use the method `dispose()`.

**Returns:**

a new [XMLValidator](#) instance of this AltovaXMLFactory.

**getXQuery**

```
public XQuery getXQuery()
```

Retrieves the XQuery engine. The object's connection to the engine must be released after use. To do this, use the method `dispose()`.

**Returns:**

a new [XQuery 1.0 engine](#) instance of this AltovaXMLFactory.

**getXSLT1**

```
public XSLT1 getXSLT1()
```

Retrieves the XSLT 1.0 engine. The object's connection to the engine must be released after use. To do this, use the method `dispose()`.

**Returns:**

a new [XSLT 1.0 engine](#) instance of this AltovaXMLFactory.

**getXSLT2**

```
public XSLT2 getXSLT2()
```

Retrieves the XSLT 2.0 engine. The object's connection to the engine must be released after use. To do this, use the method `dispose()`.

**Returns:**

a new [XSLT 2.0 engine](#) instance of this AltovaXMLFactory.

**XMLValidator**

```
public class XMLValidator  
extends java.lang.Object
```

**Description**

Class holding XMLValidator. No direct construction/access possible. Access the XML Validator by calling the function [getXMLValidator\(\)](#) on an instance of [AltovaXMLFactory](#).

**Methods**

The following methods are defined.

**getLastErrorMessage**

```
public java.lang.String getLastErrorMessage()
```

Gets the last error message from the engine.

**Returns:**

a string containing the last error message.

**isValid**

```
public boolean isValid()
```

Validates the input XML data against the DTD/Schema specified in it.

**Returns:**

true on success, false on failure. In case of failure, you can use the method `getLastErrorMessage()`.

**isValidWithExternalSchemaOrDTD**

```
public boolean isValidWithExternalSchemaOrDTD()
```

Validates the input XML data against the external DTD/Schema, which can be specified with the functions `setDTDFilename()`, `setDTDFromText()`, `setSchemaFileName()`, and `setSchemaFromText()`. *For a description of these methods, see below.*

**Returns:**

true on success, false on failure. In case of failure, you can use the method `getLastErrorMessage()`.

**isWellFormed**

```
public boolean isWellFormed()
```

Checks the input XML data for well-formedness.

**Returns:**

true on success, false on failure. In case of failure, you can use the method `getLastErrorMessage()`.

**dispose**

```
public void dispose()
```

Releases the object's connection to the COM server.

**setDTDFilename**

```
public void setDTDFilename(java.lang.String str)
```

Set file name of external DTD.

**Parameters:**

`str`: an absolute URL giving the base location of the DTD.

**setDTDFromText**

```
public void setDTDFromText(java.lang.String str)
```

Sets text value for external DTD.

**Parameters:**

`str`: a string containing DTD as text.

**setInputXMLFileName**

```
public void setInputXMLFileName(java.lang.String str)
```

Sets the file name for the input XML data. Note that you must use absolute URLs.

**Parameters:**

`str`: an absolute URL giving the base location of the XML data.

**setInputXMLFromText**

```
public void setInputXMLFromText(java.lang.String str)
```

Sets the text value for the input XML data. Example: `setInputXMLFromText( "<doc><a>text</a> </doc>" )`

**Parameters:**

`str`: a string containing XML data.

**setSchemaFileName**

```
public void setSchemaFileName( java.lang.String str)
```

Set file name of external Schema.

**Parameters:**

`str`: an absolute URL giving the base location of the Schema.

**setSchemaFromText**

```
public void setSchemaFromText( java.lang.String str)
```

Sets text value for external Schema.

**Parameters:**

`str`: a string containing Schema as text.

**setTreatXBRLInconsistenciesAsErrors**

```
public void setTreatXBRLInconsistenciesAsErrors( boolean param)
```

Sets a boolean value to treat XBRL inconsistencies as errors or not.

**Parameters:**

`param`: a Boolean value.

## XQuery

```
public class XQuery
extends java.lang.Object
```

### Description

Class holding the XQuery 1.0 engine. No direct construction/access possible. Access the XQuery Engine by calling the function [getXQuery\(\)](#) on an instance of [AltovaXMLFactory](#).

### Methods

The following methods are defined.

**addExternalVariable**

```
public void addExternalVariable( java.lang.String strName,
                                java.lang.String strVal)
```

Adds the name and value of an external variable.

**Parameters:**

`strName`: a string containing a valid QName as the variable name.

`strVal`: a string containing the value of the variable; this value will be used as a string.

**addExternalVariableAsXPath**

```
public void addExternalVariableAsXPath( java.lang.String strName,
                                         java.lang.String strVal)
```

Add name and value for an external variable, with value being evaluated as an XPath 2.0

expression.

**Parameters:**

**strName:** a string containing a valid QName as the variable name.

**strVal:** a string containing the value of the variable; the value will be evaluated as an XPath 2.0 expression.

**clearExternalVariableList**

public void **clearExternalVariableList**()

Clear the list of external variables.

**dispose**

public void **dispose**()

Releases the object's connection to the COM server.

**execute**

public void **execute**(java.lang.String sOutFile)

Executes and saves the result to file. In case of an error, you can use the method

`getLastErrorMessage()`.

**Parameters:**

**sOutFile:** an absolute URL giving the location of the output file.

**Returns:**

true on

*Note:* If no output parameter is specified, then the return type is void:

public void **execute**(java.lang.String sOutFile)

**executeAndGetResultAsString**

public java.lang.String **executeAndGetResultAsString**()

Executes and returns the result as a UTF-16 text string. In case of an error, you can use the method `getLastErrorMessage()`.

**Returns:**

string containing the serialized result. On error, will return the empty string.

**getLastErrorMessage**

public java.lang.String **getLastErrorMessage**()

Gets the last error message from the engine.

**Returns:**

a string containing the last error message.

**getOutputEncoding**

public java.lang.String **getOutputEncoding**()

Retrieves the encoding specified for the result document.

**Returns:**

a string containing the encoding name.

**getOutputIndent**

public boolean **getOutputIndent**()

Retrieves the output indent option specified for the result document.

Returns:  
the current value of the indent serialization parameter.

**getOutputMethod**

public java.lang.String **getOutputMethod**()  
Retrieves the serialization method for the result document.

Returns:  
the current serialization method.

**getOutputOmitXMLDeclaration**

public boolean **getOutputOmitXMLDeclaration**()  
Retrieves the value of omitXMLDeclaration option specified for the result document.

Returns:  
boolean value of the omit-xml-declaration parameter.

**setDotNetExtensionsEnabled**

public void **enableDotNetExtensions**(boolean bEnable)  
Enable/disable .NET extension functions.

**setInputXMLFileName**

public void **setInputXMLFileName**(java.lang.String str)  
Sets the file name for the input XML data. Note that you must use absolute URLs.

Parameters:  
str: an absolute URL giving the base location of the XML data.

**setInputXMLFromText**

public void **setInputXMLFromText**(java.lang.String str)  
Sets the text value for the input XML data. Example: setInputXMLFromText( "<doc><a>text</a> </doc>" ).

Parameters:  
str: a string containing XML data.

**setJavaExtensionsEnabled**

public void **enableJavaExtensions**(boolean bEnable)  
Enable/disable Java extension functions.

**setOutputEncoding**

public void **setOutputEncoding**(java.lang.String str)  
Sets the encoding for the result document.

Parameters:  
str: a string containing an encoding name (for example: UTF-8, UTF-16, ASCII, 8859-1, 1252 )

**setOutputIndent**

public void **setOutputIndent**(boolean bVal)  
Enables/disables the indentation option for the result document.

**Parameters:**

bVal: boolean value to enable/disable indentation.

**setOutputMethod**

```
public void setOutputMethod(java.lang.String str)
```

Sets the serialization method for the result document.

**Parameters:**

str: a string containing the serialization method. Valid values: xml, xhtml, html, text.

**setOutputOmitXMLDeclaration**

```
public void setOutputOmitXMLDeclaration(boolean bVal)
```

Enables/disables the serialization option `omitXMLDeclaration` for the result document.

**Parameters:**

bVal: a new boolean value for the `omit-xml-declaration` parameter.

**setXQueryFileName**

```
public void setXQueryFileName(java.lang.String str)
```

Sets file name of the XQuery document.

**Parameters:**

str: an absolute URL giving the base location of the XQuery file.

**setXQueryFromText**

```
public void setXQueryFromText(java.lang.String str)
```

Sets the text value for the XQuery statement.

**Parameters:**

str: a string containing the XQuery statement.

**XSLT1**

```
public class XSLT1  
extends java.lang.Object
```

**Description**

Class holding the XSLT 1.0 engine. No direct construction/access possible. Access the XSLT 1.0 Engine by calling the function [getXSLT1\(\)](#) on an instance of [AltovaXMLFactory](#).

**Methods**

The following methods are defined.

**addExternalParameter**

```
public void addExternalParameter(java.lang.String strName,  
java.lang.String strVal)
```

Adds the name and value of an external parameter.

**Parameters:**

strName: a string containing a valid QName as the parameter name.

strVal: a string containing the value of the parameter; this value will be evaluated as an XPath expression.



**clearExternalParameterList**

```
public void clearExternalParameterList()
```

Clears the list of external parameters.

**dispose**

```
public void dispose()
```

Releases the object's connection to the COM server.

**execute**

```
public void execute(java.lang.String sOutFile)
```

Executes and saves the result to file. In case of an error, you can use the method `getLastErrorMessage()`.

**Parameters:**

`sOutFile`: an absolute URL giving the location of the output file.

**Returns:**

true on success, false on error.

*Note:* If no output parameter is specified, then the return type is void:

```
public void execute(java.lang.String sOutFile)
```

**executeAndGetResultAsString**

```
public java.lang.String executeAndGetResultAsString()
```

Executes and returns the result as a UTF-16 text string. In case of an error, you can use the method `getLastErrorMessage()`.

**Returns:**

string containing the serialized result. On error, will return the empty string.

**getLastErrorMessage**

```
public java.lang.String getLastErrorMessage()
```

Gets the last error message from the engine.

**Returns:**

a string containing the last error message.

**setDotNetExtensionsEnabled**

```
public void enableDotNetExtensions(boolean bEnable)
```

Enable/disable .NET extension functions.

**setInputXMLFileName**

```
public void setInputXMLFileName(java.lang.String str)
```

Sets the file name for the input XML data. Note that you have to use absolute URLs.

**Parameters:**

`str`: an absolute URL giving the base location of the XML data.

**setInputXMLFromText**

```
public void setInputXMLFromText(java.lang.String str)
```

Sets the text value for the input XML data. Example: `setInputXMLFromText( "<doc><a>text</a> </doc>" )`.

**Parameters:**

`str`: a string containing XML data.

**setJavaExtensionsEnabled**

```
public void enableJavaExtensions(boolean bEnable)
```

Enable/disable Java extension functions.

**setXSLFileName**

```
public void setXSLTFileName(java.lang.String str)
```

Sets the file name for the XSLT data.

**Parameters:**

`str`: an absolute URL giving the base location of the XSLT data

**setXSLFromText**

```
public void setXSLTFromText(java.lang.String str)
```

Sets the text value for the XSLT data.

**Parameters:**

`str`: a string containing serialized XSLT data.

**setXSLStackSize**

```
public void setXSLTStackSize(long nVal)
```

The stack size is the maximum depth of executed instructions. If the stack size is exceeded during a transformation, an error is reported.

**Parameters:**

`nVal`: numeric value for new stack size. Must be greater than 100. The initial value 1000.

**XSLT2**

```
public class XSLT2  
extends java.lang.Object
```

**Description**

Class holding the XSLT 2.0 engine. No direct construction/access possible. Access the XSLT 2.0 Engine by calling the function [getXSLT2\(\)](#) on an instance of [AltovaXMLFactory](#). Note that the XSLT 2.0 Engine can be used in its backward compatibility mode to process an XSLT 1.0 stylesheet. The output, however, could be different than that produced by the XSLT 1.0 Engine processing the same XSLT 1.0 stylesheet.

**Methods**

The following methods are defined.

**addExternalParameter**

```
public void addExternalParameter(java.lang.String strName,  
java.lang.String strVal)
```

Adds the name and value of an external parameter.

**Parameters:**

`strName`: a string containing a valid QName as the parameter name.

`strVal`: a string containing the value of the parameter; this value will be evaluated as an XPath expression.

**clearExternalParameterList**

```
public void clearExternalParameterList()
```

Clears the list of external parameters.

**dispose**

```
public void dispose()
```

Releases the object's connection to the COM server.

**execute**

```
public void execute(java.lang.String sOutFile)
```

Executes and saves the result to file. In case of an error, you can use the method

```
getLastErrorMessage().
```

Parameters:

`sOutFile`: an absolute URL giving the location of the output file.

Returns:

true on success, false on error.

*Note:* If no output parameter is specified, then the return type is void:

```
public void execute(java.lang.String sOutFile)
```

**executeAndGetResultAsString**

```
public java.lang.String executeAndGetResultAsString()
```

Executes and returns the result as a UTF-16 text string. In case of an error, you can use the method `getLastErrorMessage()`.

Returns:

string containing the serialized result. On error, will return the empty string.

**getLastErrorMessage**

```
public java.lang.String getLastErrorMessage()
```

Gets the last error message from the engine.

Returns:

a string containing the last error message.

**setDotNetExtensionsEnabled**

```
public void enableDotNetExtensions(boolean bEnable)
```

Enable/disable .NET extension functions.

**setInitialTemplateMode**

```
public void setInitialTemplateMode(java.lang.String str)
```

Sets the initial template mode for the transformation.

**setInitialTemplateName**

```
public void setInitialTemplateName(java.lang.String str)
```

Sets the initial template name for the transformation.

**setInputXMLFileName**

```
public void setInputXMLFileName(java.lang.String str)
```

Sets the file name for the input XML data. Note that you have to use absolute URLs.

**Parameters:**

`str`: an absolute URL giving the base location of the XML data.

**setInputXMLFromText**

```
public void setInputXMLFromText(java.lang.String str)
```

Sets the text value for the input XML data. Example: `setInputXMLFromText( "<doc><a>text</a> </doc>" )`.

**Parameters:**

`str`: a string containing XML data.

**setJavaExtensionsEnabled**

```
public void enableJavaExtensions(boolean bEnable)
```

Enable/disable Java extension functions.

**setXSLFileName**

```
public void setXSLTFileName(java.lang.String str)
```

Sets the file name for the XSLT data.

**Parameters:**

`str`: an absolute URL giving the base location of the XSLT data

**setXSLFromText**

```
public void setXSLTFromText(java.lang.String str)
```

Sets the text value for the XSLT data.

**Parameters:**

`str`: a string containing serialized XSLT data.

**setXSLStackSize**

```
public void setXSLTStackSize(long nVal)
```

The stack size is the maximum depth of executed instructions. If the stack size is exceeded during a transformation, an error is reported.

**Parameters:**

`nVal`: numeric value for new stack size. Must be greater than 100. The initial value 1000.

### 2.3.3 Old Java API (Obsolete)

The objects described in this section (Old Java API) are obsolete from v2012 onwards.

For information about how to access the Application API from Java code, see the section: [Java Interface](#).

The AltovaXML Java interface (`AltovaXML.jar`) connects to the AltovaXML COM interface using native functions in the `AltovaXMLLib.dll`. This DLL will have been installed in the `WINDIR\system32\` directory when you install AltovaXML using the AltovaXML installer. `AltovaXML.jar` contains the package `com.altova.engines`, which is the package containing the Altova engines.

#### Setup

In order to use the Java interface, add the `AltovaXML.jar` file to the `CLASSPATH`. COM registration is done automatically by the AltovaXML Installer. If you change the location of the file `AltovaXML_COM.exe` after installation, you should register AltovaXML as a COM server object by running the command `AltovaXML_COM.exe /regserver`. See [Registering AltovaXML as a COM Server Object](#) for more details.

#### Documentation

This section contains a detailed description of the AltovaXML Java interface. This documentation is also available in HTML format in the ZIP archive, `AltovaXMLJavaDocs.zip`, which is located in the `AltovaXML2012` application folder.

#### Examples

For detailed examples, see the example files in the `AltovaXMLExamples` folder in the application folder.

#### The `com.altova.engines` package

To use the Java interface, your starting point is the package `com.altova.engines`. This is the Java interface for the AltovaXML COM server object; it provides access to XMLValidator and to the XSLT 1.0, XSLT 2.0 and XQuery 1.0 engines.

The `com.altova.engines` package provides connection to the AltovaXML COM interface using the native functions in `AltovaXMLLib.dll`, which is installed in the `WINDIR\system32\` directory.

To connect to a new instance of AltovaXML COM server object, use the static method `getInstance()` of the `AltovaXMLFactory` class. From the returned interface you can choose the required engine using the `getENGINEINSTANCEInstance()` function.

Given below is a sample of code that uses the Java interface:

```
import com.altova.engines.*;
```

```

/**
 * Test application for AltovaXML COM components java interface
 */
public class AltovaXMLTest {
    /**
     * public constructor for AltovaXMLTest
     */
    public AltovaXMLTest(){
    }

    /**
     * application main
     */
    public static void main(String[] args) {
        System.out.println("AltovaXML Java Interface Test Application");

        //request a COM server object - fails if AltovaXML is not registered
        IAltovaXMLFactory objXmlApp = AltovaXMLFactory.getInstance();

        if ( objXmlApp != null ) {
            //get interface for the XQuery engine
            IXQuery xquery = objXmlApp.getXQueryInstance();
            //set XQuery statement
            xquery.setXQueryStatement("<doc><a>{1 to 3}</a>This data is
well-formed.</doc>");
            //execute the statement previously set.
            //There was no input XML specified so the initial context is
empty.
            String sres = xquery.executeAndGetResultAsString();
            //release XQuery engine's connection to the COM server object
            xquery.releaseInstance();
            System.out.println(sres);

            IXMLValidator validator = objXmlApp.getXMLValidatorInstance();
            validator.setInputXMLFromText(sres);
            boolean b = validator.isWellFormed();
            if ( b )
                System.out.println("XML data is well-formed.");
            else
                System.out.println("Data is not well-formed.");
            validator.releaseInstance();

            //release Application object connection to the COM server object.
            //After this the COM server object will shut down automatically.
            objXmlApp.releaseInstance();
        } else{
            System.out.println("Creating instance of IAltovaXMLFactory
failed.");
            System.out.println("Please make sure AltovaXML.exe is correctly
registered!");
        }
    }
}

```

### Example

The code listing below checks whether the submitted XML file is well-formed, and then executes an XQuery document.

To connect to a new instance of AltovaXML COM server object, use the static method `getInstance()` of the `AltovaXMLFactory` class. From the returned interface you can choose the required engine using the `getENGINENameInstance()` function (for example:

```

getXMLValidatorInstance() ).

        // Locate samples installed with the product.
        // REMARK: You will need to modify this if you use a different
major version.
String strExamplesFolder = System.getenv( "ProgramFiles" ) +
"/Altova/AltovaXML2011/AltovaXMLExamples/";

String inFilename = strExamplesFolder + "simple.xml";
String xqFilename = strExamplesFolder + "CopyInput.xq";
System.out.println( "AltovaXML Java JNI XQuery" );

    try
    {
        // get application instance
        IAltovaXMLFactory objXmlApp = AltovaXMLFactory.getInstance
( );

        // get XML Validator and XQ method pointers from the
application instance
        IXMLValidator validator =
objXmlApp.getXMLValidatorInstance();
        IXQuery xQuery = objXmlApp.getXQueryInstance();

        // remove comments on line below to see error being caught*/
        validator.setInputXMLFileName( inFilename );
        if ( validator.isWellFormed() )
        {
            // if the file is well-formed copy it using XQuery
            xQuery.setInputXMLFileName( inFilename );
            xQuery.setXQueryFileName( xqFilename );

            // test return value
            String resultString =
xQuery.executeAndGetResultAsString();
            if ( resultString == null )
                System.out.println( "XQuery error: " +
xQuery.getLastErrorMessage() );
            else
                System.out.println( "Transform contents: " +
resultString );
        }
        else
            System.out.println( "Not wellformed error: " +
validator.getLastErrorMessage() );

        // release instance pointer
        objXmlApp.releaseInstance();
    }
    catch ( Exception e)
    {
        System.out.println( "Error: " + e );
    }
}

```

### Examples

For more examples, see the example files in the `AltovaXMLExamples` folder of the AltovaXML application folder.

## Interfaces

Given below is a summary of the interfaces of `com.altova.engines`. Detailed descriptions are given in the respective sections.

- [IAltovaXMLEngine](#)  
Basic interface for XMLValidator, and XSLT 1.0, XSLT 2.0, and XQuery 1.0 engines.
- [IAltovaXMLFactory](#)  
Interface for AltovaXML COM object wrapper.
- [IExecutable](#)  
Executable interface for engines.
- [IReleasable](#)  
Interface for Release functionality.
- [IXMLValidator](#)  
Interface for XMLValidator.
- [IXQuery](#)  
Interface for the XQuery 1.0 engine.
- [IXSLT](#)  
Interface for the XSLT engines.

### IAltovaXMLEngine

Basic interface for XMLValidator, XSLT 1.0, XSLT 2.0 and XQuery engines. Public interface that extends [IReleasable](#).

**Superinterface:** [IReleasable](#)

**Subinterface:** [XMLValidator](#), [IXQuery](#), [IXSLT](#)

**Implementing classes:** [XMLValidator](#), [XQuery](#), [XSLT1](#), [XSLT2](#)

### Methods

The following methods are defined.

#### **setInputXMLFileName**

```
public void setInputXMLFileName(java.lang.String filename)
```

Sets the file name for the input XML data. Please note that you have to use absolute URLs.

**Parameters:**

`filename`: an absolute URL giving the base location of the XML data.

#### **setInputXMLFromText**

```
public void setInputXMLFromText(java.lang.String text)
```

Sets the text value for the input XML data. For example: `setInputXMLFromText( "<doc><a>text</a> </doc>" )`

**Parameters:**

`text`: a string containing XML data.

#### **getLastErrorMessage**

```
public java.lang.String getLastErrorMessage()
```

Gets the last error message from the engine.

**Returns:**

a string containing the last error message.



## IAltovaXMLFactory

Interface for AltovaXML COM object wrapper. Provides access to the interfaces of XMLValidator, XSLT 1.0, XSLT 2.0 and Xquery 1.0 engines. Public interface that extends [IReleasable](#).

**Superinterface:** [IReleasable](#)

**Implementing classes:** [AltovaXMLFactory](#)

### Methods

The following methods are defined.

#### **getXQueryInstance**

```
public IXQuery getXQueryInstance()
```

Creates a new instance of XQuery class for the current XQuery engine instance. The object's connection to the engine must be released after use. To do this, use the function [releaseInstance\(\)](#) declared in the [IReleasable](#) interface.

**Returns:**

the [IXQuery](#) interface of the newly created class.

#### **getXSLT1Instance**

```
public IXSLT getXSLT1Instance()
```

Creates a new instance of XSLT1 class for the current XSLT 1.0 engine instance. The object's connection to the engine must be released after use. To do this, use the function [releaseInstance\(\)](#) declared in the [IReleasable](#) interface.

**Returns:**

the [IXSLT](#) interface of the newly created class.

#### **getXSLT2Instance**

```
public IXSLT getXSLT2Instance()
```

Creates a new instance of XSLT2 class for the current XSLT 2.0 engine instance. The object's connection to the engine must be released after use. To do this, use the function [releaseInstance\(\)](#) declared in the [IReleasable](#) interface.

**Returns:**

the [IXSLT](#) interface of the newly created class.

#### **getXMLValidatorInstance**

```
public IXMLValidator getXMLValidatorInstance()
```

Creates a new instance of XMLValidator class for the current XML Validator instance. The object's connection to the engine must be released after use. To do this, use the function [releaseInstance\(\)](#) declared in the [IReleasable](#) interface.

**Returns:**

the [IXMLValidator](#) interface of the newly created class.

The following methods enable the addition of catalogs used for document lookup. After catalogs are added they are used for lookup till the COM server terminates. Added catalogs cannot be removed.

```
app.AddXMLCatalogDefault()  
Adds Altova's default RootCatalog.xml to the catalogs  
  
app.AddXMLCatalogFromFile( string catalogfilename )  
Adds the catalog identified by catalogfilename to the catalogs  
  
app.AddXMLCatalogFromText( string catalogtext )  
Adds the catalog with content catalogtext to the catalogs
```

## IExecutable

Executable interface for engines. Public interface.

**Subinterface:** [IXQuery](#), [IXSLT](#)

**Implementing classes:** [XQuery](#), [XSLT1](#), [XSLT2](#)

## Methods

The following methods are defined.

### execute

```
public boolean execute( java.lang.String outfilename )  
Executes and saves the result to file. In case of an error, you can use the function  
getLastErrorMessage\(\) declared in IAltovaXMLEngine to get additional information.
```

#### Parameters:

outfilename: an absolute URL giving the location of the output file.

#### Returns:

true on success, false on error.

### executeAndGetResultAsString

```
public java.lang.String executeAndGetResultAsString()  
Executes and returns the result as string. In case of an error, you can use the function  
getLastErrorMessage\(\) declared in IAltovaXMLEngine to get additional information.
```

#### Returns:

string containing the serialized result. On error, will return the empty string.

### enableJavaExtensions

```
public void enableJavaExtensions( boolean bEnable )  
Enables/disables .NET extension functions.
```

### enableDotNetExtensions

```
public void enableDotNetExtensions( boolean bEnable )  
Enables/disables Java extension functions.
```

## IReleasable

Public interface for Release functionality. When an object implementing this interface is not used any more, then the `releaseInstance()` function must be called in order to release connection to the COM server. The COM server will shut down automatically when all connections to it are released.

**Subinterface:** [IXQuery](#), [IXSLT](#)  
**Implementing classes:** [XQuery](#), [XSLT1](#), [XSLT2](#)

### Methods

The following methods are defined.

#### **releaseInstance**

```
public void releaseInstance()
```

Releases the object's connection to the COM server.

### IXMLValidator

Interface for the XML Validator. Public interface that extends [IAltovaXMLEngine](#).

**Superinterface:** [IAltovaXMLEngine](#), [IReleasable](#)  
**Implementing classes:** [XMLValidator](#)

### Methods

The following methods are defined.

#### **isValid**

```
public boolean isValid()
```

Validates the input XML data against the DTD/Schema specified in it.

#### **Returns:**

true on success, false on failure. In case of failure, you can use the function [getLastErrorMessage\(\)](#) declared in [IAltovaXMLEngine](#) to get additional information.

#### **isWellFormed**

```
public boolean isWellFormed()
```

Checks the input XML data for well-formedness.

#### **Returns:**

true on success, false on failure. In case of failure, you can use the function [getLastErrorMessage\(\)](#) declared in [IAltovaXMLEngine](#) to get additional information.

#### **isValidWithExternalSchemaOrDTD**

```
public boolean isValidWithExternalSchemaOrDTD()
```

Validates the input XML data against the external DTD/Schema which can be specified with the functions [setDTDFileName\(\)](#), [setDTDFromText\(\)](#), [setSchemaFileName\(\)](#), [setSchemaFromText\(\)](#).

#### **Returns:**

true on success, false on failure. In case of failure, you can use the function [getLastErrorMessage\(\)](#) declared in [IAltovaXMLEngine](#) to get additional information.

#### **setSchemaFileName**

```
public void setSchemaFileName(java.lang.String filename)
```

Sets file name for external Schema.

#### **Parameters:**

filename: an absolute URL giving the base location of the Schema

**setDTDFileName**

```
public void setDTDFileName(java.lang.String filename)
```

Sets file name for external DTD.

**Parameters:**

filename: an absolute URL giving the base location of the DTD.

**setSchemaFromText**

```
public void setSchemaFromText(java.lang.String text)
```

Sets text value for external Schema.

**Parameters:**

text: string containing Schema as text.

**setDTDFromText**

```
public void setDTDFromText(java.lang.String text)
```

Sets text value for external DTD.

**Parameters:**

text: string containing DTD as text.

**TreatXBRLInconsistenciesAsErrors**

```
public void TreatXBRLInconsistenciesAsErrors(boolean bEnable)
```

If set to `True`, returns XBRL semantic inconsistencies as errors. Default is `False`.

**Parameters:**

bEnable: boolean

**IXQuery**

Interface for the XQuery engine. Public interface that extends [IAltovaXMLEngine](#) and

[IExecutable](#).

**Superinterface:** [IAltovaXMLEngine](#), [IExecutable](#), [IReleasable](#)

**Implementing classes:** [XQuery](#)

**Methods**

The following methods are defined.

**setXQueryFileName**

```
public void setXQueryFileName(java.lang.String filename)
```

Sets the file name of the XQuery document.

**Parameters:**

filename: an absolute URL giving the base location of the XQuery file.

**setXQueryStatement**

```
public void setXQueryStatement(java.lang.String text)
```

Sets the text value of the XQuery statement.

**Parameters:**

text: a string containing the XQuery statement.

**setOutputEncoding**

```
public void setOutputEncoding(java.lang.String encoding)
```

Sets the encoding of the result document.

**Parameters:**

encoding: a string containing the name of the encoding name (for example: UTF-8, UTF-16, ASCII, 8859-1, 1252).

**getOutputEncoding**

```
public java.lang.String getOutputEncoding()
```

Retrieves the encoding specified for the result document.

**Returns:**

a string containing an encoding name.

**setOutputIndent**

```
public void setOutputIndent(boolean indent)
```

Enables/disables the indentation option for the result document.

**Parameters:**

indent: boolean value to enable/disable output indentation.

**getOutputIndent**

```
public boolean getOutputIndent()
```

Retrieves the output indent option specified for the result document.

**Returns:**

boolean value indicating whether output is indented (`true`) or not (`false`).

**setOutputMethod**

```
public void setOutputMethod(java.lang.String method)
```

Sets the serialization method for the result document.

**Parameters:**

method: a string containing the serialization method. (Valid values are: xml, xhtml, html, text).

**getOutputMethod**

```
public java.lang.String getOutputMethod()
```

Retrieves the serialization method for the result document.

**Returns:**

a string containing the serialization method for the output document.

**setOutputOmitXMLDeclaration**

```
public void setOutputOmitXMLDeclaration(boolean decl)
```

Enables/disables the serialization option `omitXMLDeclaration` for the result document.

**Parameters:**

decl: new boolean value for the `omit-xml-declaration` parameter.

**getOutputOmitXMLDeclaration**

```
public boolean getOutputOmitXMLDeclaration()
```

Retrieve the value of `omitXMLDeclaration` option specified for the result document.

**Returns:**

boolean value indicating whether output document contains an XML declaration (`true`) or not (

false).

**addExternalVariable**

```
public void addExternalVariable(java.lang.String name,  
                               java.lang.String val)
```

Add name and value for an external variable.

**Parameters:**

**name:** a string containing a valid QName as the variable name.

**val:** a string containing the value of the variable; the value will be used as a string.

**addExternalVariableAsXPath**

```
public void addExternalVariableAsXPath(java.lang.String name,  
                                       java.lang.String val)
```

Add name and value for an external variable, with value being evaluated as an XPath 2.0 expression.

**Parameters:**

**name:** a string containing a valid QName as the variable name.

**val:** a string containing the value of the variable; the value will be evaluated as an XPath 2.0 expression.

**clearExternalVariableList**

```
public void clearExternalVariableList()
```

Clears the list of external variables.

**IXSLT**

Interface for the XSLT engines. Public interface that extends [IAltovaXMLEngine](#) and [IExecutable](#).

**Superinterface:** [IAltovaXMLEngine](#), [IExecutable](#), [IReleasable](#)

**Implementing classes:** [XSLT1](#) and [XSLT2](#)

**Note:** The XSLT 2.0 Engine can be used in its backward compatibility mode to process an XSLT 1.0 stylesheet. The output, however, could be different than that produced by the XSLT 1.0 Engine processing the same XSLT 1.0 stylesheet.

**Methods**

The following methods are defined.

**setXSLTFileName**

```
public void setXSLTFileName(java.lang.String name)
```

Sets the file name for the XSLT data.

**Parameters:**

**name:** an absolute URL giving the base location of the XSLT data file.

**setXSLTFromText**

```
public void setXSLTFromText(java.lang.String text)
```

Sets text value for the XSLT data.

**Parameters:**

**text:** a string containing serialized XSLT data.

**addExternalParameter**

```
public void addExternalParameter(java.lang.String name,  
                                java.lang.String val)
```

Adds the name and value of an external parameter.

**Parameters:**

**name:** a string containing a valid QName as the parameter name.

**val:** a string containing the value of the parameter; the value will be evaluated as an XPath expression.

**clearExternalParameterList**

```
public void clearExternalParameterList()
```

Clears the list of external parameters.

**setXSLTStackSize**

```
public void setXSLTStackSize(long nVal)
```

The stack size is the maximum depth of executed instructions. If the stack size is exceeded during a transformation, an error is reported.

**Parameters:**

**nVal:** numeric value for new stack size. Must be greater than 100. The initial value 1000.

**Classes**

Given below is a summary of the classes of `com.altova.engines`. Detailed descriptions are given in the respective sections.

- [AltovaXMLFactory](#)  
Creates new AltovaXML COM server object instance via native call, and provides access to AltovaXML engines.
- [XMLValidator](#)  
Class holding XMLValidator.
- [XQuery](#)  
Class holding the XQuery 1.0 Engine.
- [XSLT1](#)  
Class holding the XSLT 1.0 Engine.
- [XSLT2](#)  
Class holding the XSLT 2.0 Engine.

**AltovaXMLFactory**

```
public class AltovaXMLFactory  
extends java.lang.Object  
implements IAltovaXMLFactory
```

**Implemented interfaces:** [IAltovaXMLFactory](#), [IReleasable](#)

**Description**

Creates new AltovaXML COM server object instance via native call, and provides access to the AltovaXML engines. The relationship between `AltovaXMLFactory` and the AltovaXML COM object is one-to-one. This means that subsequent calls to the `getENGINENameInstance()` function will return interfaces for the same engine instance.

## Methods

The following methods are defined.

### **getInstance**

```
public static IAltovaXMLFactory getInstance()
```

Creates a new [AltovaXMLFactory](#) object and connects it to a new [AltovaXML](#) COM server object.

#### Returns:

the interface [IAltovaXMLFactory](#) for the newly created [AltovaXMLFactory](#) object or null if the creation of the COM object failed. In the latter case you should make sure that [AltovaXML.exe](#) is [properly registered](#) as a COM server object.

### **releaseInstance**

```
public void releaseInstance()
```

Releases the object's connection to the COM server.

#### Specified by:

[releaseInstance](#) in interface [IReleasable](#).

### **getXQueryInstance**

```
public IXQuery getXQueryInstance()
```

Creates a new instance of [XQuery](#) class for the current [XQuery](#) engine instance. The object's connection to the engine must be released after use. To do this, use the function [releaseInstance\(\)](#) declared in the [IReleasable](#) interface.

#### Specified by:

[getXQueryInstance](#) in interface [IAltovaXMLFactory](#).

#### Returns:

the [IXQuery](#) interface of the newly created class.

### **getXSLT1Instance**

```
public IXSLT getXSLT1Instance()
```

Creates a new instance of [XSLT1](#) class for the current [XSLT 1.0](#) engine instance. The object's connection to the engine must be released after use. To do this, use the function [releaseInstance\(\)](#) declared in the [IReleasable](#) interface.

#### Specified by:

[getXSLT1Instance](#) in interface [IAltovaXMLFactory](#).

#### Returns:

the [IXSLT](#) interface of the newly created class.

### **getXSLT2Instance**

```
public IXSLT getXSLT2Instance()
```

Creates a new instance of [XSLT2](#) class for the current [XSLT 2.0](#) engine instance. The object's connection to the engine must be released after use. To do this, use the function [releaseInstance\(\)](#) declared in the [IReleasable](#) interface.

#### Specified by:

[getXSLT2Instance](#) in interface [IAltovaXMLFactory](#).

#### Returns:

the [IXSLT](#) interface of the newly created class.



**getXMLValidatorInstance**

```
public IXMLValidator getXMLValidator()
```

Creates a new instance of [XMLValidator](#) class for the current XML Validator instance. The object's connection to the engine must be released after use. To do this, use the function [releaseInstance\(\)](#) declared in the [IReleasable](#) interface.

**Specified by:**

[getXMLValidatorInstance](#) in interface [IAltovaXMLFactory](#).

**Returns:**

the [IXMLValidator](#) interface of the newly created class.

The following methods enable the addition of catalogs used for document lookup. After catalogs are added they are used for lookup till the COM server terminates. Added catalogs cannot be removed.

```
app. AddXMLCatalogDefault()
```

Adds Altova's default `RootCatalog.xml` to the catalogs

```
app. AddXMLCatalogFromFile( string catalogfilename )
```

Adds the catalog identified by `catalogfilename` to the catalogs

```
app. AddXMLCatalogFromText( string catalogtext )
```

Adds the catalog with content `catalogtext` to the catalogs

**XMLValidator**

```
public class XMLValidator  
extends java.lang.Object  
implements IXMLValidator
```

**Implemented interfaces:** [IAltovaXMLEngine](#), [IReleasable](#), [IXMLValidator](#)

**Description**

Class holding XMLValidator. No direct construction/access possible. Get the [IXMLValidator](#) interface to it by calling the function [getXMLValidatorInstance\(\)](#) on an instance of [IAltovaXMLFactory](#).

**Constructors**

The following constructor is defined.

**XMLValidator**

```
protected XMLValidator(long nValidatorPtr)
```

**Methods**

The following methods are defined.

**releaseInstance**

```
public void releaseInstance()
```

Releases the object's connection to the COM server.

**Specified by:**

[releaseInstance](#) in interface [IReleasable](#).

**setInputXMLFileName**

```
public void setInputXMLFileName(java.lang.String str)
```

Sets the file name for the input XML data. Note that you must use absolute URLs.

Specified by:

[setInputXMLFileName](#) in interface [IAltovaXMLEngine](#).

Parameters:

str: an absolute URL giving the base location of the XML data.

**setInputXMLFromText**

```
public void setInputXMLFromText(java.lang.String str)
```

Sets the text value for the input XML data. Example: `setInputXMLFromText( "<doc><a>text</a> </doc>" )`

Specified by:

[setInputXMLFromText](#) in interface [IAltovaXMLEngine](#).

Parameters:

str: a string containing XML data.

**getLastErrorMessage**

```
public java.lang.String getLastErrorMessage()
```

Gets the last error message from the engine.

Specified by:

[getLastErrorMessage](#) in interface [IAltovaXMLEngine](#).

Returns:

a string containing the last error message.

**isValid**

```
public boolean isValid()
```

Validates the input XML data against the DTD/Schema specified in it.

Specified by:

[isValid](#) in interface [IXMLValidator](#).

Returns:

true on success, false on failure. In case of failure, you can use the function [getLastErrorMessage](#) declared in [IAltovaXMLEngine](#) to get additional information.

**isWellFormed**

```
public boolean isWellFormed()
```

Checks the input XML data for well-formedness.

Specified by:

[isWellFormed](#) in interface [IXMLValidator](#).

Returns:

true on success, false on failure. In case of failure, you can use the function [getLastErrorMessage](#) declared in [IAltovaXMLEngine](#) to get additional information.

**isValidWithExternalSchemaOrDTD**

```
public boolean isValidWithExternalSchemaOrDTD()
```

Validates the input XML data against the external DTD/Schema, which can be specified with the functions `setDTDFileName()`, `setDTDFromText()`, `setSchemaFileName()`, and

`setSchemaFromText()` . *For a description of these functions, see below.*

**Specified by:**

[isValidWithExternalSchemaOrDTD](#) in interface [IXMLValidator](#).

**Returns:**

`true` on success, `false` on failure. In case of failure, you can use the function [getLastErrorMessage](#) declared in [IAltovaXMLEngine](#) to get additional information.

#### **setSchemaFileName**

```
public void setSchemaFileName(java.lang.String str)
```

Set file name of external Schema.

**Specified by:**

[setSchemaFileName](#) in interface [IXMLValidator](#).

**Parameters:**

`str`: an absolute URL giving the base location of the Schema.

#### **setDTDFileName**

```
public void setDTDFileName(java.lang.String str)
```

Set file name of external DTD.

**Specified by:**

[setDTDFileName](#) in interface [IXMLValidator](#).

**Parameters:**

`str`: an absolute URL giving the base location of the DTD.

#### **setSchemaFromText**

```
public void setSchemaFromText(java.lang.String str)
```

Sets text value for external Schema.

**Specified by:**

[setSchemaFromText](#) in interface [IXMLValidator](#).

**Parameters:**

`str`: a string containing Schema as text.

#### **setDTDFromText**

```
public void setDTDFromText(java.lang.String str)
```

Sets text value for external DTD.

**Specified by:**

[setDTDFromText](#) in interface [IXMLValidator](#).

**Parameters:**

`str`: a string containing DTD as text.

#### **XQuery**

```
public class XQuery
extends java.lang.Object
implements IXQuery
```

**Implemented interfaces:** [IAltovaXMLEngine](#), [IExecutable](#), [IReleasable](#), [IXQuery](#)

#### **Description**

Class holding the XQuery 1.0 engine. No direct construction/access possible. Get the [IXQuery](#)

interface to it by calling the function [getXQueryInstance\(\)](#) on an instance of [IAltovaXMLFactory](#).

### Constructors

The following constructor is defined.

#### XQuery

```
protected XQuery( long nXQueryPtr)
```

### Methods

The following methods are defined.

#### releaseInstance

```
public void releaseInstance()
```

Releases the object's connection to the COM server.

Specified by:

[releaseInstance](#) in interface [IReleasable](#).

#### execute

```
public boolean execute( java.lang.String sOutFile)
```

Executes and saves the result to file. In case of an error, you can use the function [getLastErrorMessage\(\)](#) declared in [IAltovaXMLEngine](#) to get additional information.

Specified by:

[execute](#) in interface [IExecutable](#).

Parameters:

sOutFile: an absolute URL giving the location of the output file.

Returns:

true on success, false on error.

#### executeAndGetResultAsString

```
public java.lang.String executeAndGetResultAsString()
```

Executes and returns the result as a UTF-16 text string. In case of an error, you can use the function [getLastErrorMessage\(\)](#) declared in [IAltovaXMLEngine](#) to get additional information.

Specified by:

[executeAndGetResultAsString](#) in interface [IExecutable](#).

Returns:

string containing the serialized result. On error, will return the empty string.

#### setInputXMLFileName

```
public void setInputXMLFileName( java.lang.String str)
```

Sets the file name for the input XML data. Note that you must use absolute URLs.

Specified by:

[setInputXMLFileName](#) in interface [IAltovaXMLEngine](#).

Parameters:

str: an absolute URL giving the base location of the XML data.

#### setInputXMLFromText

```
public void setInputXMLFromText(java.lang.String str)  
Sets the text value for the input XML data. Example: setInputXMLFromText( "<doc>  
<a>text</a> </doc>" ).
```

Specified by:

[setInputXMLFromText](#) in interface [IAltovaXMLEngine](#).

Parameters:

str: a string containing XML data.

#### **getLastErrorMessage**

```
public java.lang.String getLastErrorMessage()
```

Gets the last error message from the engine.

Specified by:

[getLastErrorMessage](#) in interface [IAltovaXMLEngine](#).

Returns:

a string containing the last error message.

#### **setXQueryFileName**

```
public void setXQueryFileName(java.lang.String str)
```

Sets file name of the XQuery document.

Specified by:

[setXQueryFileName](#) in interface [IXQuery](#).

Parameters:

str: an absolute URL giving the base location of the XQuery file.

#### **setXQueryStatement**

```
public void setXQueryStatement(java.lang.String str)
```

Sets the text value for the XQuery statement.

Specified by:

[setXQueryStatement](#) in interface [IXQuery](#)

Parameters:

str: a string containing the XQuery statement.

#### **setOutputEncoding**

```
public void setOutputEncoding(java.lang.String str)
```

Sets the encoding for the result document.

Specified by:

[setOutputEncoding](#) in interface [IXQuery](#).

Parameters:

str: a string containing an encoding name (for example: UTF-8, UTF-16, ASCII, 8859-1, 1252 )

#### **getOutputEncoding**

```
public java.lang.String getOutputEncoding()
```

Retrieves the encoding specified for the result document.

Specified by:

[getOutputEncoding](#) in interface [IXQuery](#).

Returns:

a string containing the encoding name.

**setOutputIndent**

```
public void setOutputIndent(boolean bVal)
```

Enables/disables the indentation option for the result document.

Specified by:

[setOutputIndent](#) in interface [IXQuery](#).

Parameters:

bVal: boolean value to enable/disable indentation.

**getOutputIndent**

```
public boolean getOutputIndent()
```

Retrieves the output indent option specified for the result document.

Specified by:

[getOutputIndent](#) in interface [IXQuery](#).

Returns:

the current value of the indent serialization parameter.

**setOutputMethod**

```
public void setOutputMethod(java.lang.String str)
```

Sets the serialization method for the result document.

Specified by:

[setOutputMethod](#) in interface [IXQuery](#).

Parameters:

str: a string containing the serialization method. Valid values: xml, xhtml, html, text.

**getOutputMethod**

```
public java.lang.String getOutputMethod()
```

Retrieves the serialization method for the result document.

Specified by:

[getOutputMethod](#) in interface [IXQuery](#).

Returns:

the current serialization method.

**setOutputOmitXMLDeclaration**

```
public void setOutputOmitXMLDeclaration(boolean bVal)
```

Enables/disables the serialization option `omitXMLDeclaration` for the result document.

Specified by:

[setOutputOmitXMLDeclaration](#) in interface [IXQuery](#).

Parameters:

bVal: a new boolean value for the `omit-xml-declaration` parameter.

**getOutputOmitXMLDeclaration**

```
public boolean getOutputOmitXMLDeclaration()
```

Retrieves the value of `omitXMLDeclaration` option specified for the result document.

Specified by:

[getOutputOmitXMLDeclaration](#) in interface [IXQuery](#).

Returns:

boolean value of the `omit-xml-declaration` parameter.

**addExternalVariable**

```
public void addExternalVariable(java.lang.String strName,  
                               java.lang.String strVal)
```

Adds the name and value of an external variable.

Specified by:

[addExternalVariable](#) in interface [IXQuery](#).

Parameters:

`strName`: a string containing a valid QName as the variable name.

`strVal`: a string containing the value of the variable; this value will be used as a string.

**addExternalVariableAsXPath**

```
public void addExternalVariableAsXPath(java.lang.String strName,  
                                       java.lang.String strVal)
```

Add name and value for an external variable, with value being evaluated as an XPath 2.0 expression.

Specified by:

[addExternalVariableAsXPath](#) in interface [IXQuery](#).

Parameters:

`strName`: a string containing a valid QName as the variable name.

`strVal`: a string containing the value of the variable; the value will be evaluated as an XPath 2.0 expression.

**clearExternalVariableList**

```
public void clearExternalVariableList()
```

Clear the list of external variables.

Specified by:

[clearExternalVariableList](#) in interface [IXQuery](#).

**enableJavaExtensions**

```
public void enableJavaExtensions(boolean bEnable)
```

Enable/disable Java extension functions.

Specified by:

[enableJavaExtensions](#) in interface [IExecutable](#).

**enableDotNetExtensions**

```
public void enableDotNetExtensions(boolean bEnable)
```

Enable/disable .NET extension functions.

Specified by:

[enableJavaExtensions](#) in interface [IExecutable](#).

**XSLT1**

```
public class XSLT1  
extends java.lang.Object  
implements IXSLT
```

**Implemented interfaces:** [IAltovaXMLEngine](#), [IExecutable](#), [IReleasable](#), [IXSLT](#)

**Description**

Class holding the XSLT 1.0 engine. No direct construction/access possible. Get the [IXSLT](#) interface to it by calling the function [getXSLT1Instance\(\)](#) on an instance of [IAltovaXMLFactory](#).

### Constructors

The following constructor is defined.

#### XSLT1

```
protected XSLT1(long nXSLT1Ptr)
```

### Methods

The following methods are defined.

#### releaseInstance

```
public void releaseInstance()
```

Releases the object's connection to the COM server.

Specified by:

[releaseInstance](#) in interface [IReleasable](#).

#### execute

```
public boolean execute(java.lang.String sOutFile)
```

Executes and saves the result to file. In case of an error, you can use the function [getLastErrorMessage\(\)](#) declared in [IAltovaXMLEngine](#) to get additional information.

Specified by:

[execute](#) in interface [IExecutable](#).

Parameters:

sOutFile: an absolute URL giving the location of the output file.

Returns:

true on success, false on error.

#### executeAndGetResultAsString

```
public java.lang.String executeAndGetResultAsString()
```

Executes and returns the result as a UTF-16 text string. In case of an error, you can use the function [getLastErrorMessage\(\)](#) declared in [IAltovaXMLEngine](#) to get additional information.

Specified by:

[executeAndGetResultAsString](#) in interface [IExecutable](#).

Returns:

string containing the serialized result. On error, will return the empty string.

#### setInputXMLFileName

```
public void setInputXMLFileName(java.lang.String str)
```

Sets the file name for the input XML data. Note that you have to use absolute URLs.

Specified by:

[setInputXMLFileName](#) in interface [IAltovaXMLEngine](#).

Parameters:

str: an absolute URL giving the base location of the XML data.

#### setInputXMLFromText



```
public void setInputXMLFromText(java.lang.String str)
Sets the text value for the input XML data. Example: setInputXMLFromText( "<doc>
<a>text</a> </doc>" ).
```

Specified by:

[setInputXMLFromText](#) in interface [IAltovaXMLEngine](#).

Parameters:

str: a string containing XML data.

#### **getLastErrorMessage**

```
public java.lang.String getLastErrorMessage()
```

Gets the last error message from the engine.

Specified by:

[getLastErrorMessage](#) in interface [IAltovaXMLEngine](#).

Returns:

a string containing the last error message.

#### **setXSLTFileName**

```
public void setXSLTFileName(java.lang.String str)
```

Sets the file name for the XSLT data.

Specified by:

[setXSLTFileName](#) in interface [IXSLT](#).

Parameters:

str: an absolute URL giving the base location of the XSLT data

#### **setXSLTFromText**

```
public void setXSLTFromText(java.lang.String str)
```

Sets the text value for the XSLT data.

Specified by:

[setXSLTFromText](#) in interface [IXSLT](#).

Parameters:

str: a string containing serialized XSLT data.

#### **addExternalParameter**

```
public void addExternalParameter(java.lang.String strName,
                                 java.lang.String strVal)
```

Adds the name and value of an external parameter.

Specified by:

[addExternalParameter](#) in interface [IXSLT](#).

Parameters:

strName: a string containing a valid QName as the parameter name.

strVal: a string containing the value of the parameter; this value will be evaluated as an XPath expression.

#### **clearExternalParameterList**

```
public void clearExternalParameterList()
```

Clears the list of external parameters.

Specified by:

[clearExternalParameterList](#) in interface [IXSLT](#).

**setXSLTStackSize**

```
public void setXSLTStackSize(long nVal)
```

The stack size is the maximum depth of executed instructions. If the stack size is exceeded during a transformation, an error is reported.

Specified by:

[setXSLTStackSize](#) in interface [IXSLT](#).

Parameters:

nVal: numeric value for new stack size. Must be greater than 100. The initial value 1000.

**enableJavaExtensions**

```
public void enableJavaExtensions(boolean bEnable)
```

Enable/disable Java extension functions.

Specified by:

[enableJavaExtensions](#) in interface [IExecutable](#).

**enableDotNetExtensions**

```
public void enableDotNetExtensions(boolean bEnable)
```

Enable/disable .NET extension functions.

Specified by:

[enableJavaExtensions](#) in interface [IExecutable](#).

**XSLT2**

```
public class XSLT2  
extends java.lang.Object  
implements IXSLT
```

**Implemented interfaces:** [IAltovaXMLEngine](#), [IExecutable](#), [IReleasable](#), [IXSLT](#)

**Description**

Class holding the XSLT 2.0 engine. No direct construction/access possible. Get the [IXSLT](#) interface to it by calling the function [getXSLT2Instance\(\)](#) on an instance of [IAltovaXMLFactory](#). Note that the XSLT 2.0 Engine can be used in its backward compatibility mode to process an XSLT 1.0 stylesheet. The output, however, could be different than that produced by the XSLT 1.0 Engine processing the same XSLT 1.0 stylesheet.

**Constructors**

The following constructor is defined.

**XSLT2**

```
protected XSLT2(long nXSLT2Ptr)
```

**Methods**

The following methods are defined.

dispose

**releaseInstance**

```
public void releaseInstance()
```

Releases the object's connection to the COM server.

Specified by:

[releaseInstance](#) in interface [IReleasable](#).

#### **execute**

```
public boolean execute(java.lang.String sOutFile)
```

Executes and saves the result to file. In case of an error, you can use the function [getLastErrorMessage\(\)](#) declared in [IAltovaXMLEngine](#) to get additional information.

**Specified by:**

[execute](#) in interface [IExecutable](#).

**Parameters:**

sOutFile: an absolute URL giving the location of the output file.

**Returns:**

true on success, false on error.

#### **executeAndGetResultAsString**

```
public java.lang.String executeAndGetResultAsString()
```

Executes and returns the result as a UTF-16 text string. In case of an error, you can use the function [getLastErrorMessage\(\)](#) declared in [IAltovaXMLEngine](#) to get additional information.

**Specified by:**

[executeAndGetResultAsString](#) in interface [IExecutable](#).

**Returns:**

string containing the serialized result. On error, will return the empty string.

#### **setInputXMLFileName**

```
public void setInputXMLFileName(java.lang.String str)
```

Sets the file name for the input XML data. Note that you have to use absolute URLs.

**Specified by:**

[setInputXMLFileName](#) in interface [IAltovaXMLEngine](#).

**Parameters:**

str: an absolute URL giving the base location of the XML data.

#### **setInputXMLFromText**

```
public void setInputXMLFromText(java.lang.String str)
```

Sets the text value for the input XML data. Example: `setInputXMLFromText( "<doc><a>text</a> </doc>" )`.

**Specified by:**

[setInputXMLFromText](#) in interface [IAltovaXMLEngine](#).

**Parameters:**

str: a string containing XML data.

#### **getLastErrorMessage**

```
public java.lang.String getLastErrorMessage()
```

Gets the last error message from the engine.

**Specified by:**

[getLastErrorMessage](#) in interface [IAltovaXMLEngine](#).

**Returns:**

a string containing the last error message.

#### **setXSLTFileName**

```
public void setXSLTFileName(java.lang.String str)
```

Sets the file name for the XSLT data.

Specified by:

[setXSLTFileName](#) in interface [IXSLT](#).

Parameters:

str: an absolute URL giving the base location of the XSLT data

#### **setXSLTFromText**

```
public void setXSLTFromText(java.lang.String str)
```

Sets the text value for the XSLT data.

Specified by:

[setXSLTFromText](#) in interface [IXSLT](#).

Parameters:

str: a string containing serialized XSLT data.

#### **addExternalParameter**

```
public void addExternalParameter(java.lang.String strName,  
                                 java.lang.String strVal)
```

Adds the name and value of an external parameter.

Specified by:

[addExternalParameter](#) in interface [IXSLT](#).

Parameters:

strName: a string containing a valid QName as the parameter name.

strVal: a string containing the value of the parameter; this value will be evaluated as an XPath expression.

#### **clearExternalParameterList**

```
public void clearExternalParameterList()
```

Clears the list of external parameters.

Specified by:

[clearExternalParameterList](#) in interface [IXSLT](#).

#### **setInitialTemplateName**

```
public void setInitialTemplateName(java.lang.String str)
```

Sets the initial template name for the transformation.

#### **setInitialTemplateMode**

```
public void setInitialTemplateMode(java.lang.String str)
```

Sets the initial template mode for the transformation.

#### **setXSLTStackSize**

```
public void setXSLTStackSize(long nVal)
```

The stack size is the maximum depth of executed instructions. If the stack size is exceeded during a transformation, an error is reported.

Specified by:

[setXSLTStackSize](#) in interface [IXSLT](#).

Parameters:

nVal: numeric value for new stack size. Must be greater than 100. The initial value 1000.

**enableJavaExtensions**

```
public void enableJavaExtensions(boolean bEnable)
```

Enable/disable Java extension functions.

Specified by:

[enableJavaExtensions](#) in interface [IExecutable](#).

**enableDotNetExtensions**

```
public void enableDotNetExtensions(boolean bEnable)
```

Enable/disable .NET extension functions.

Specified by:

[enableJavaExtensions](#) in interface [IExecutable](#).

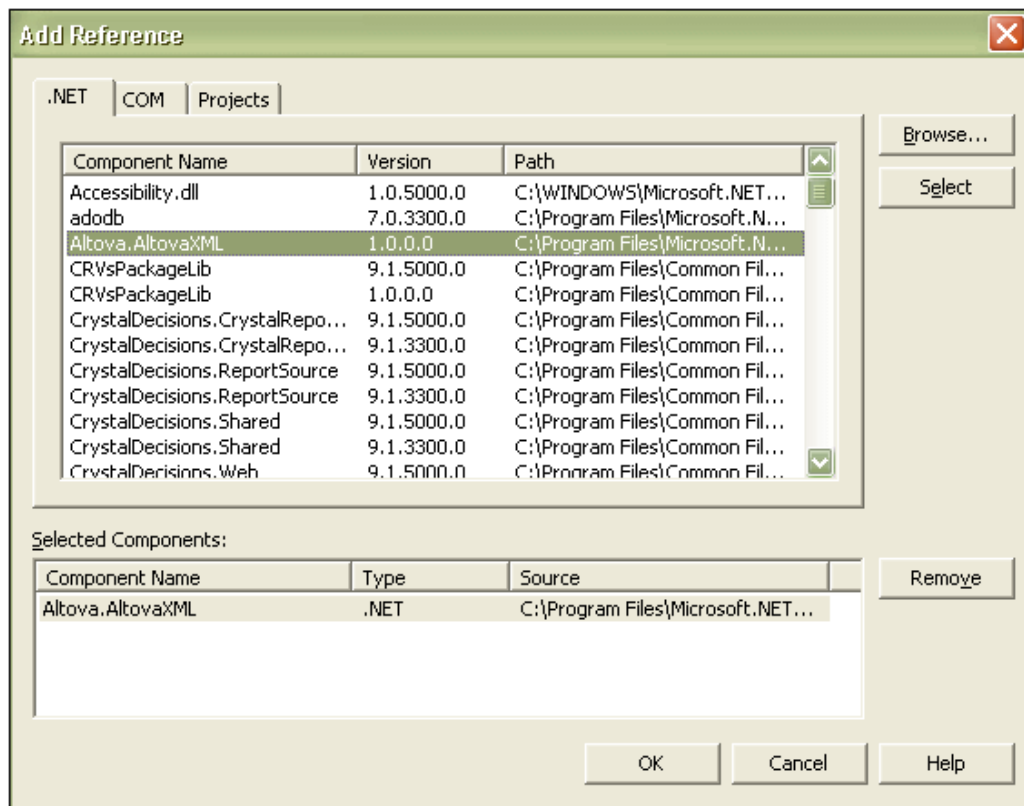
## 2.4 .NET Interface

The .NET interface is built as a wrapper around the AltovaXML COM interface. It is provided as a primary interop assembly signed by Altova and using the namespace `Altova.AltovaXML`. In order to use AltovaXML in your .NET project, you need to: (i) add a reference to the AltovaXML DLL (which is called `Altova.AltovaXML.dll`) in your project, and (ii) have AltovaXML registered as a COM server object. Once these requirements (which are described below) have been met, you can use the AltovaXML functionality in your project.

### Adding the AltovaXML DLL as a reference to the project

The AltovaXML package contains a signed DLL file, named `Altova.AltovaXML.dll`, which will automatically be added to the global assembly cache (and the .NET reference library) when AltovaXML is installed using the AltovaXML installer. (It will be located typically in the `C:\WINDOWS\assembly` folder.) To add this DLL as a reference in a .NET project, do the following:

1. With the .NET project open, click **Project | Add Reference**. The Add Reference dialog (screenshot below) pops up, displaying a list of installed .NET components. (Note: If the AltovaXML component is not in the .NET tab list, it can be selected from the COM tab.)



2. Select `Altova.AltovaXML` from the component list, double-click it or press the **Select** button, then click **OK**.

### Registering AltovaXML as a COM server object

COM registration is done automatically by the AltovaXML Installer. If you change the location of the file `AltovaXML_COM.exe` after installation, you should register AltovaXML as a COM server object by running the command `AltovaXML_COM.exe /regserver`. (Note that the correct path to the `AltovaXML_COM.exe` must be entered. See [Registering AltovaXML as a COM Server](#))

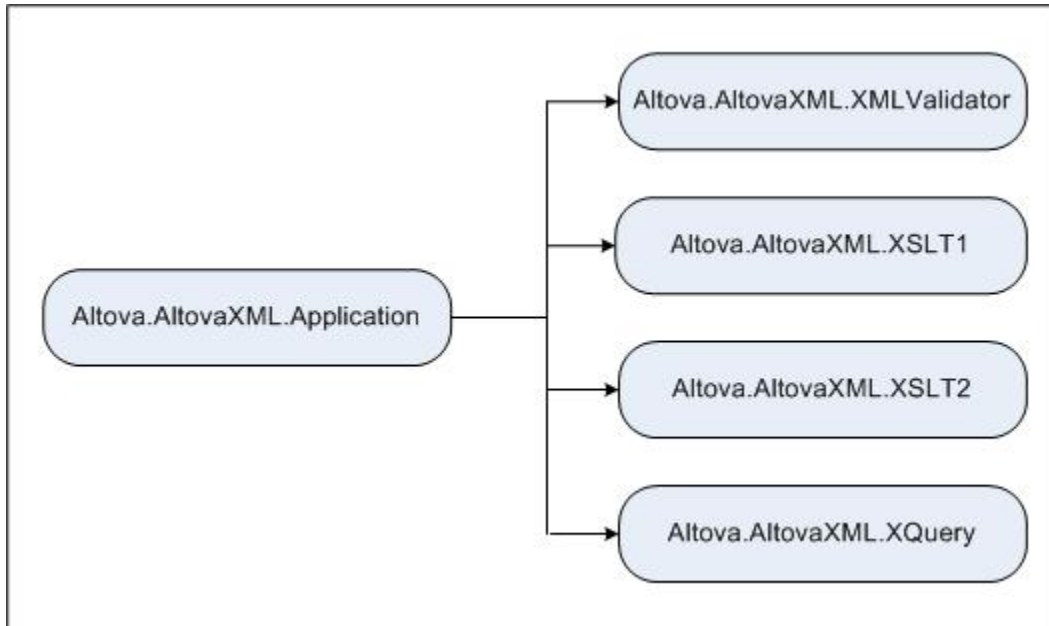
[Object](#) for more details.)

Once the `Altova.AltovaXML.dll` is available to the .NET interface and AltovaXML has been registered as a COM server object, AltovaXML functionality will be available in your .NET project.

**Note:** If you receive an access error, check that permissions are correctly set. Go to Component Services and give permissions to the same account that runs the application pool containing AltovaXML.

### 2.4.1 General Usage

The classes and methods you can use are as described in the [COM Interface](#) section, but are in the namespace `Altova.AltovaXML`. They are listed in the following sections. The starting point is the `Altova.AltovaXML.Application` object. When you create this object, a connection to a new AltovaXML COM server object is created. The object model is shown in the diagram below.



#### Methods

The following methods, which are available on the application object, enable the addition of catalogs used for document lookup. After catalogs are added they are used for lookup till the COM server terminates. Added catalogs cannot be removed.

```
app.AddXMLCatalogDefault()
Adds Altova's default RootCatalog.xml to the catalogs
```

```
app.AddXMLCatalogFromFile( string catalogfilename )
Adds the catalog identified by catalogfilename to the catalogs
```

```
app.AddXMLCatalogFromText( string catalogtext )
Adds the catalog with content catalogtext to the catalogs
```

#### Example

How to use the AltovaXML classes and methods in the .NET framework is shown in the C# code for a button event listed below. A [fuller example](#) is given at the end of the .NET Interface section.

```
private void button1_Click(object sender, System.EventArgs e)
{
    Altova.AltovaXML.ApplicationClass appXML = new
Altova.AltovaXML.ApplicationClass();
    Altova.AltovaXML.XMLValidator XMLValidator = appXML.XMLValidator;
    XMLValidator.InputXMLFromText = "<test>Is this data well-formed?
<a></test>";
}
```



```
if ( XMLValidator.IsWellFormed() )
{
    MessageBox.Show( this, "The input data is well-formed" ) ;
}
else
{
    MessageBox.Show( this, "The input data is not well-formed" ) ;
}
}
```

The code listing above does the following:

1. The `Altova.AltovaXML.ApplicationClass` object is created, which creates a connection to a new `AltovaXML` COM server object.
2. The XML Validator functionality is called using `Altova.AltovaXML.XMLValidator`.
3. The `InputXMLFromText` property of `Altova.AltovaXML.XMLValidator` submits the input XML data.
4. The `IsWellFormed` method of `Altova.AltovaXML.XMLValidator` checks whether the submitted XML data is well-formed, returning `TRUE` or `FALSE`.

For more detailed examples, see the example files in the `AltovaXMLExamples` folder in the application folder.

## 2.4.2 Altova.AltovaXML.XMLValidator

### Description

The `Altova.AltovaXML.XMLValidator` object provides methods to test:

- The well-formedness of an XML document.
- The validity of an XML document against a DTD or XML Schema referenced from within the XML document.
- The validity of an XML document against a DTD or XML Schema supplied externally via the code.
- The validity of an XBRL document against an XBRL taxonomy (a `.xsd` file).

All these methods return Boolean `TRUE` or `FALSE`.

**Note:** Where string inputs are to be interpreted as URLs, absolute paths should be used. If a relative path is used, a mechanism to resolve the relative path should be defined in the calling module.

### Methods

The following methods are available:

**IsWellFormed()** as Boolean

`IsWellFormed()` checks the well-formedness of the XML document. Returns `TRUE` if the XML document is well-formed, `FALSE` if it is not well-formed.

**IsValid()** as Boolean

`IsValid` validates the XML document against the DTD or XML Schema referenced in the XML document. Returns `TRUE` if the XML document is valid, `FALSE` if invalid. To validate against a DTD or XML Schema not referenced in the XML document, use the method `IsValidWithExternalSchemaOrDTD`.

**IsValidWithExternalSchemaOrDTD()** as Boolean

`IsValidWithExternalSchemaOrDTD` validates the XML document against the DTD or XML Schema supplied by any one of the following properties: `SchemaFileName`, `DTDFileName`, `SchemaFromText`, or `DTDFromText`. If more than one of these properties has values set for it, then the `IsValidWithExternalSchemaOrDTD` method uses the property that has been set last. Returns `TRUE` if the XML document is valid, `FALSE` if invalid. To validate against a DTD or XML Schema referenced in the XML document, use the method `IsValid`.

**Note:** Validation and well-formedness checks must always occur after assigning the XML and/or DTD or XML Schema document to the respective properties.

### Properties

The following properties are defined:

**InputXMLFileName**

A string input that is read as a URL to locate the XML file to be validated.

**SchemaFileName**

A string input that is read as a URL to locate the XML Schema file against which the XML document is to be validated.

**DTDFileName**

A string input that is read as a URL to locate the DTD file against which the XML document is to be validated.

**InputXMLFromText**

A string input that constructs an XML document.

**SchemaFromText**

A string input that constructs an XML Schema document.

**DTDFromText**

A string input that constructs a DTD document.

**LastErrorMessage**

Returns the last error message.

**TreatXBRLInconsistenciesAsErrors**

If set to `True`, returns XBRL semantic inconsistencies as errors. Default is `False`.

**Example**

The following C# code snippet shows how to validate an XML document. A [fuller example](#) is given at the end of the .NET Interface section.

To create these code snippet in a C# project, do the following:

1. In Microsoft Visual Studio, add a new project using **File | New | Project**.
2. Add a reference to the AltovaXML DLL by clicking **Project | Add Reference**. The Add Reference dialog pops up, displaying a list of installed .NET components. Select the AltovaXML component from the list to add it. (Note: If the AltovaXML component is not in the .NET tab list, it can be selected from the COM tab.)
3. Enter the example code snippet below in the project form. The code snippet below validates an XML file. The XML file used in this code snippet is located in the `AltovaXMLExamples` folder of the AltovaXML application folder.
4. Compile the code and test it.

```
// Locate examples installed with AltovaXML
// REMARK: You might need to adapt this if you have a different major version
// of the product (2011 in this example).
String strExamplesFolder = Environment.GetEnvironmentVariable
("ProgramFiles") + "\\Altova\\AltovaXML2011\\AltovaXMLExamples\\";

// Create a new AltovaXML instance and access its engines
Altova.AltovaXML.Application AltovaXML = new Altova.AltovaXML.Application
();

// Validate input file simple.xml - it must be well-formed but not necessarily
// valid.
// The AltovaXML application will provide us with a validator object.
Altova.AltovaXML.XMLValidator AltovaXMLValidator = AltovaXML.XMLValidator;
AltovaXMLValidator.InputXMLFileName = strExamplesFolder + "simple.xml";
Boolean bIsWellFormed = AltovaXMLValidator.IsWellFormed();
Boolean bIsValid = AltovaXMLValidator.IsValid();

// Show result
MessageBox.Show("File " + strExamplesFolder + "simple.xml" + " is " +
    (bIsWellFormed ? "well-formed" : "not Well-formed") +
    " and " + (bIsValid ? "valid" : "invalid") + ".");
```

### 2.4.3 Altova.AltovaXML.XSLT1

#### Description

The `Altova.AltovaXML.XSLT1` object provides methods and properties to execute an XSLT 1.0 transformation using the Altova XSLT 1.0 Engine. Results can be saved to a file or returned as a string. The object also enables XSLT parameters to be passed to the XSLT stylesheet. The URLs of XML and XSLT files can be supplied as strings via the object's properties. Alternatively, the XML and XSLT documents can be constructed within the code as text strings.

**Note:** Where string inputs are to be interpreted as URLs, absolute paths should be used. If a relative path is used, a mechanism to resolve the relative path should be defined in the calling module.

#### Methods

The following methods are available:

##### **Execute**(OutputFileName as String)

void **execute**(String outputFilename)

`Execute` executes an XSLT 1.0 transformation and saves the result to an output file, the name and location of which is provided as an input string to the `Execute` method. For example:

```
Execute("C:\OutputDoc.xml").
```

##### **ExecuteAndGetResultAsString**( ) as String

`ExecuteAndGetResultAsString` executes an XSLT 1.0 transformation and returns the result as a UTF-16 text string. *See below for examples.*

##### **AddExternalParameter**(ParamName as String, ParamValue as String)

Takes a parameter name and the value of this parameter as input arguments. Each external parameter and its value is to be specified in a separate call to the method. If multiple calls specify the same parameter name the value set by the latest will be used. Since parameter values are XPath expressions, parameter values that are strings must be enclosed in single quotes. In this example, two parameter values are submitted:

```
AddExternalParameter("Param1", "' http://www.altova.com/'");  
AddExternalParameter("Param2", "concat(' http://www.altova.com/',  
MyFile/@url)");
```

##### **ClearExternalParameterList**( )

No argument should be provided. The `ClearExternalParameterList` clears the external parameters list created with `AddExternalParameter` methods.

**Note:** Transformation must always occur after assigning the XML and XSLT documents.

#### Properties

The following properties are defined:

##### **InputXMLFileName**

A string input that is read as a URL to locate the XML file to be transformed.

##### **XSLFileName**

A string input that is read as a URL to locate the XSLT file to be used for the transformation.

##### **InputXMLFromText**

A string input that constructs an XML document.

##### **XSLFromText**

A string input that constructs an XSLT document.

**XSLStackSize**

The stack size is the maximum depth of executed instructions. The stack size can be changed with the `XSLStackSize` property. The minimum allowed stack size is 100. The default stack size is 1000. If the stack size is exceeded during a transformation, an error is reported.

**LastErrorMessage**

Returns the last error message.

**JavaExtensionsEnabled**

Enables Java extensions. You can specify whether Java extensions should be enabled or not by submitting `true` or `false` (case-insensitive) as a `Variant_Bool` argument.

**DotNetExtensionsEnabled**

Enables .NET extensions. You can specify whether .NET extensions should be enabled or not by submitting `true` or `false` (case-insensitive) as a `Variant_Bool` argument.

**Examples**

The following C# code snippets show how to:

- Validate an XML document and run an XSLT 1.0 transformation (transformation from XML file to string)
- Transform using XSLT 1.0 (XML file to XML file)
- Transform using XSLT 1.0 (string to XML file)
- Transform using XSLT 1.0 (string to string)

A [fuller example](#) is given at the end of the .NET Interface section.

To create these code snippet in a C# project, do the following:

1. In Microsoft Visual Studio, add a new project using **File | New | Project**.
2. Add a reference to the AltovaXML DLL by clicking **Project | Add Reference**. The Add Reference dialog pops up, displaying a list of installed .NET components. Select the AltovaXML component from the list to add it. (Note: If the AltovaXML component is not in the .NET tab list, it can be selected from the COM tab.)
3. Enter the example code snippet below in the project form. The code snippet below validates an XML file and runs an XSLT 1.0 transformation on the XML file. The files used in this code snippet are located in the `AltovaXMLExamples` folder of the AltovaXML application folder.
4. Compile the code and test it.

**Validation and XSLT 1.0 transformation (XML to String)**

```
// Specify folder (AltovaXMLExamples folder)
// Check if filepath is correct for you
String strExamplesFolder = Environment.GetEnvironmentVariable
("ProgramFiles") + "\\Altova\\AltovaXML2011\\AltovaXMLExamples\\";

// Create a new AltovaXML instance and access its engines
Altova.AltovaXML.Application AltovaXML = new Altova.AltovaXML.Application
();

// Use Validator of AltovaXML to validate input file simple.xml
// File must be well-formed but not necessarily valid
Altova.AltovaXML.XMLValidator AltovaXMLValidator = AltovaXML.XMLValidator;
AltovaXMLValidator.InputXMLFileName = strExamplesFolder + "simple.xml";
Boolean bIsWellFormed = AltovaXMLValidator.IsWellFormed();
```

```

    Boolean bIsValid = AltovaXMLValidator.IsValid();

    // Show result
    MessageBox.Show("File " + strExamplesFolder + "simple.xml" + " is " +
        (bIsWellFormed ? "well-formed" : "not Well-formed") +
        " and " + (bIsValid ? "valid" : "invalid") + ".");

    if (bIsWellFormed)
    {
        // Use XSLT1 Engine of AltovaXML to transform simple.xml using
        CopyInputXSLT1.xsl
        Altova.AltovaXML.IXSLT1 AltovaXMLXSLT1 = AltovaXML.XSLT1;
        AltovaXMLXSLT1.InputXMLFileName = strExamplesFolder + "simple.xml";
        AltovaXMLXSLT1.XSLFileName = strExamplesFolder + "CopyInputXSLT1.
xsl";
        String strResult = AltovaXMLXSLT1.ExecuteAndGetResultAsString();

        // Show result
        MessageBox.Show("XSLT 1.0 engine answered: " + strResult);
    }

```

### **XSLT 1.0 transformation (XML to XML)**

```

// Specify folder (AltovaXMLExamples folder)
// Check if filepath is correct for you
String strExamplesFolder = Environment.GetEnvironmentVariable
("ProgramFiles") + "\\Altova\\AltovaXML2011\\AltovaXMLExamples\\";

// Create a new AltovaXML instance and access its engines
Altova.AltovaXML.Application AltovaXML = new Altova.AltovaXML.Application
();

// Use XSLT1 Engine of AltovaXML to transform simple.xml using CopyInputXSLT1.
xsl
Altova.AltovaXML.IXSLT1 AltovaXMLXSLT1 = AltovaXML.XSLT1;
AltovaXMLXSLT1.InputXMLFileName = strExamplesFolder + "simple.xml";
AltovaXMLXSLT1.XSLFileName = strExamplesFolder + "CopyInputXSLT1.xsl";
AltovaXMLXSLT1.Execute(strExamplesFolder + "simpleOutputFromXML.xml");

```

### **XSLT 1.0 transformation (String to XML)**

```

// Specify folder (AltovaXMLExamples folder)
// Check if filepath is correct for you
String strExamplesFolder = Environment.GetEnvironmentVariable
("ProgramFiles") + "\\Altova\\AltovaXML2011\\AltovaXMLExamples\\";

// Create a new AltovaXML instance and access its engines
Altova.AltovaXML.Application AltovaXML = new Altova.AltovaXML.Application
();

// Use XSLT1 Engine of AltovaXML to transform input string using
CopyInputXSLT1.xsl
Altova.AltovaXML.IXSLT1 AltovaXMLXSLT1 = AltovaXML.XSLT1;
AltovaXMLXSLT1.InputXMLFromText = "<?xml version='1.0'?><doc>Hello
World</doc>";
AltovaXMLXSLT1.XSLFileName = strExamplesFolder + "CopyInputXSLT1.xsl";
AltovaXMLXSLT1.Execute(strExamplesFolder + "simpleOutputFromString.xml");

```

### **XSLT 1.0 transformation (String to String)**

```

// Specify folder (AltovaXMLExamples folder)
// Check if filepath is correct for you
String strExamplesFolder = Environment.GetEnvironmentVariable
("ProgramFiles") + "\\Altova\\AltovaXML2011\\AltovaXMLExamples\\";

// Create a new AltovaXML instance and access its engines
Altova.AltovaXML.Application AltovaXML = new Altova.AltovaXML.Application
();

// Use XSLT1 Engine of AltovaXML to transform input string using
CopyInputXSLT1.xsl
Altova.AltovaXML.IXSLT1 AltovaXMLXSLT1 = AltovaXML.XSLT1;
AltovaXMLXSLT1.InputXMLFromText = "<?xml version='1.0'?><doc>Hello
World</doc>";
AltovaXMLXSLT1.XSLFileName = strExamplesFolder + "CopyInputXSLT1.xsl";
String strResult = AltovaXMLXSLT1.ExecuteAndGetResultAsString();

// Show result
MessageBox.Show("XSLT 1.0 engine answered: " + strResult);

```

### Using .NET extensions

```

// Specify folder (AltovaXMLExamples folder)
// Check if filepath is correct for you
String strExamplesFolder = Environment.GetEnvironmentVariable
("ProgramFiles") + "\\Altova\\AltovaXML2011\\AltovaXMLExamples\\";

// Create a new AltovaXML instance and access its engines
Altova.AltovaXML.Application AltovaXML = new Altova.AltovaXML.Application
();

// Use XSLT1 Engine from AltovaXML application
Altova.AltovaXML.IXSLT1 AltovaXMLXSLT1 = AltovaXML.XSLT1;

// Enable .NET extensions
AltovaXMLXSLT1.DotNetExtensionsEnabled = 1;

// Use XSLT containing .NET math extension for transformation
AltovaXMLXSLT1.InputXMLFileName = strExamplesFolder + "simple.xml";
AltovaXMLXSLT1.XSLFromText = "<xsl:stylesheet
xmlns:xsl='http://www.w3.org/1999/XSL/Transform'
xmlns:math='clitype: System.Math' version='1.0'><xsl:output
omit-xml-declaration='yes' />
<xsl:template match='/'><a><sqrtanswer><xsl:value-of select='
math: Sqrt(9)' /></sqrtanswer></a>
</xsl:template></xsl:stylesheet>";
AltovaXMLXSLT1.Execute(strExamplesFolder + "Output.xml");

// Release ALL references to all components that were received.
System.Runtime.InteropServices.Marshal.ReleaseComObject(AltovaXMLXSLT1);
AltovaXMLXSLT1 = null;
System.Runtime.InteropServices.Marshal.ReleaseComObject(AltovaXML);
AltovaXML = null;

```

### Using the LastErrorMessage property

```

// Specify folder (AltovaXMLExamples folder)
// Check if filepath is correct for you

```

```
String strExamplesFolder = Environment.GetEnvironmentVariable
("ProgramFiles") + "\\Altova\\AltovaXML2011\\AltovaXMLExamples\\";

// Create a new AltovaXML instance and access its engines
Altova.AltovaXML.Application AltovaXML = new Altova.AltovaXML.Application
();

// Use XSLT1 Engine from AltovaXML application
Altova.AltovaXML.IXSLT1 AltovaXMLXSLT1 = AltovaXML.XSLT1;

// Enable/disable .NET extensions (true/false, 1/0)
AltovaXMLXSLT1.DotNetExtensionsEnabled = 0;

// Use XSLT containing .NET math extension for transformation
AltovaXMLXSLT1.InputXMLFileName = strExamplesFolder + "simple.xml";
AltovaXMLXSLT1.XSLFromText = "<xsl:stylesheet
xmlns:xsl=' http://www.w3.org/1999/XSL/Transform'
xmlns:math=' clitype:System.Math' version='1.0'><xsl:output
omit-xml-declaration='yes' />
<xsl:template match='/'><a><sqrtanswer><xsl:value-of select='
math:Sqrt(9)'/></sqrtanswer></a>
</xsl:template></xsl:stylesheet>";
try
{
    AltovaXMLXSLT1.Execute(strExamplesFolder + "Output.xml");
}
catch (Exception)
{
    String strError = AltovaXMLXSLT1.LastErrorMessage;
    // Show result
    MessageBox.Show("XSLT 1.0 engine errors: " + strError);
}

// Release ALL references to all components that were received.
System.Runtime.InteropServices.Marshal.ReleaseComObject(AltovaXMLXSLT1);
AltovaXMLXSLT1 = null;
System.Runtime.InteropServices.Marshal.ReleaseComObject(AltovaXML);
AltovaXML = null;
```



## 2.4.4 Altova.AltovaXML.XSLT2

### Description

The `Altova.AltovaXML.XSLT2` object provides methods and properties to execute an XSLT 2.0 transformation using the Altova XSLT 2.0 Engine. Results can be saved to a file or returned as a string. The object also enables XSLT parameters to be passed to the XSLT stylesheet. The URLs of XML and XSLT files can be supplied as strings via the object's properties. Alternatively, the XML and XSLT documents can be constructed within the code as text strings.

### Note:

- Where string inputs are to be interpreted as URLs, absolute paths should be used. If a relative path is used, a mechanism to resolve the relative path should be defined in the calling module.
- The XSLT 2.0 Engine can be used in its backward compatibility mode to process an XSLT 1.0 stylesheet. The output, however, could be different than that produced by the XSLT 1.0 Engine processing the same XSLT 1.0 stylesheet.

### Methods

The following methods are available:

**Execute**(OutputFileName as String)

void **execute**(String outputFilename)

`Execute` executes an XSLT 2.0 transformation and saves the result to an output file, the name and location of which is provided as an input string to the `Execute` method. For example:

```
Execute("C:\OutputDoc.xml").
```

**ExecuteAndGetResultAsString**( ) as String

`ExecuteAndGetResultAsString` executes an XSLT 2.0 transformation and returns the result as a UTF-16 text string. *See below for examples.*

**AddExternalParameter**(ParamName as String, ParamValue as String)

Takes a parameter name and the value of this parameter as input arguments. Each external parameter and its value is to be specified in a separate call to the method. If multiple calls specify the same parameter name the value set by the latest will be used. Since parameter values are XPath expressions, parameter values that are strings must be enclosed in single quotes. In this example, two parameter values are submitted:

```
AddExternalParameter("Param1","' http://www.altova.com/'");  
AddExternalParameter("Param2","concat(' http://www.altova.com/', MyFile/@url)");
```

**ClearExternalParameterList**( )

No argument should be provided. The `ClearExternalParameterList` clears the external parameters list created with `AddExternalParameter` methods.

**InitialTemplateName**

Sets the initial named template. The argument is the name of the template from which processing is to start. For example: `InitialNamedTemplat="MyNamedTemplate"`.

**InitialTemplateMode**

Sets the initial mode for processing. The argument is the name of the required initial mode. Templates with this mode value will be processed. For example:

```
InitialTemplateMode="MyMode".
```

**Note:** Transformation must always occur after assigning the XML and XSLT documents.

## Properties

The following properties are defined:

### **InputXMLFileName**

A string input that is read as a URL to locate the XML file to be transformed.

### **XSLFileName**

A string input that is read as a URL to locate the XSLT file to be used for the transformation.

### **InputXMLFromText**

A string input that constructs an XML document.

### **XSLFromText**

A string input that constructs an XSLT document.

### **XSLStackSize**

The stack size is the maximum depth of executed instructions. The stack size can be changed with the `XSLStackSize` property. The minimum allowed stack size is 100. The default stack size is 1000. If the stack size is exceeded during a transformation, an error is reported.

### **LastErrorMessage**

Returns the last error message.

### **JavaExtensionsEnabled**

Enables Java extensions. You can specify whether Java extensions should be enabled or not by submitting `true` or `false` (case-insensitive) as a `VARIANT_BOOL` argument.

### **DotNetExtensionsEnabled**

Enables .NET extensions. You can specify whether .NET extensions should be enabled or not by submitting `true` or `false` (case-insensitive) as a `VARIANT_BOOL` argument.

## Examples

The following C# code snippets show how to:

- Validate an XML document and run an XSLT 2.0 transformation (transformation from XML file to string)
- Transform using XSLT 2.0 (XML file to XML file)
- Transform using XSLT 2.0 (string to XML file)
- Transform using XSLT 2.0 (string to string)

A [fuller example](#) is given at the end of the .NET Interface section.

To create these code snippet in a C# project, do the following:

1. In Microsoft Visual Studio, add a new project using **File | New | Project**.
2. Add a reference to the AltovaXML DLL by clicking **Project | Add Reference**. The Add Reference dialog pops up, displaying a list of installed .NET components. Select the AltovaXML component from the list to add it. (Note: If the AltovaXML component is not in the .NET tab list, it can be selected from the COM tab.)
3. Enter the example code snippet below in the project form. The code snippet below validates an XML file and runs an XSLT 2.0 transformation on the XML file. The files used in this code snippet are located in the `AltovaXMLExamples` folder of the AltovaXML application folder.
4. Compile the code and test it.

**Validation and XSLT 2.0 transformation (XML to string)**

```

// Specify folder (AltovaXMLExamples folder)
// Check if filepath is correct for you
String strExamplesFolder = Environment.GetEnvironmentVariable
("ProgramFiles") + "\\Altova\\AltovaXML2011\\AltovaXMLExamples\\";

// Create a new AltovaXML instance and access its engines
Altova.AltovaXML.Application AltovaXML = new Altova.AltovaXML.Application
();

// Use Validator of AltovaXML to validate input file simple.xml
// File must be well-formed but not necessarily valid
Altova.AltovaXML.XMLValidator AltovaXMLValidator = AltovaXML.XMLValidator;
AltovaXMLValidator.InputXMLFileName = strExamplesFolder + "simple.xml";
Boolean bIsWellFormed = AltovaXMLValidator.IsWellFormed();
Boolean bIsValid = AltovaXMLValidator.IsValid();

// Show result
MessageBox.Show("File " + strExamplesFolder + "simple.xml" + " is " +
    (bIsWellFormed ? "well-formed" : "not Well-formed") +
    " and " + (bIsValid ? "valid" : "invalid") + ".");

if (bIsWellFormed)
{
    // Use XSLT2 Engine of AltovaXML to transform simple.xml using
CopyInputXSLT2.xsl
    Altova.AltovaXML.IXSLT2 AltovaXMLXSLT2 = AltovaXML.XSLT2;
    AltovaXMLXSLT2.InputXMLFileName = strExamplesFolder + "simple.xml";
    AltovaXMLXSLT2.XSLFileName = strExamplesFolder + "CopyInputXSLT2.
xsl";

    String strResult = AltovaXMLXSLT2.ExecuteAndGetResultAsString();

    // Show result
    MessageBox.Show("XSLT 2.0 engine answered: " + strResult);
}

```

**XSLT 2.0 transformation (XML to XML)**

```

// Specify folder (AltovaXMLExamples folder)
// Check if filepath is correct for you
String strExamplesFolder = Environment.GetEnvironmentVariable
("ProgramFiles") + "\\Altova\\AltovaXML2011\\AltovaXMLExamples\\";

// Create a new AltovaXML instance and access its engines
Altova.AltovaXML.Application AltovaXML = new Altova.AltovaXML.Application
();

// Use XSLT2 Engine of AltovaXML to transform simple.xml using CopyInputXSLT2.
xsl
Altova.AltovaXML.IXSLT2 AltovaXMLXSLT2 = AltovaXML.XSLT2;
AltovaXMLXSLT2.InputXMLFileName = strExamplesFolder + "simple.xml";
AltovaXMLXSLT2.XSLFileName = strExamplesFolder + "CopyInputXSLT2.xsl";
AltovaXMLXSLT2.Execute(strExamplesFolder + "simpleOutputFromXML.xml");

```

**XSLT 2.0 transformation (String to XML)**

```

// Specify folder (AltovaXMLExamples folder)
// Check if filepath is correct for you

```

```

    String strExamplesFolder = Environment.GetEnvironmentVariable
("ProgramFiles") + "\\Altova\\AltovaXML2011\\AltovaXMLExamples\\";

// Create a new AltovaXML instance and access its engines
    Altova.AltovaXML.Application AltovaXML = new Altova.AltovaXML.Application
();

// Use XSLT2 Engine of AltovaXML to transform input string using
CopyInputXSLT2.xsl
    Altova.AltovaXML.IXSLT2 AltovaXMLXSLT2 = AltovaXML.XSLT2;
    AltovaXMLXSLT2.InputXMLFromText = "<?xml version='1.0' ?><doc>Hello
World</doc>";
    AltovaXMLXSLT2.XSLFileName = strExamplesFolder + "CopyInputXSLT2.xsl";
    AltovaXMLXSLT2.Execute(strExamplesFolder + "simpleOutputFromString.xml");

```

### **XSLT 2.0 transformation (String to String)**

```

// Specify folder (AltovaXMLExamples folder)
// Check if filepath is correct for you
    String strExamplesFolder = Environment.GetEnvironmentVariable
("ProgramFiles") + "\\Altova\\AltovaXML2011\\AltovaXMLExamples\\";

// Create a new AltovaXML instance and access its engines
    Altova.AltovaXML.Application AltovaXML = new Altova.AltovaXML.Application
();

// Use XSLT2 Engine of AltovaXML to transform input string using
CopyInputXSLT2.xsl
    Altova.AltovaXML.IXSLT2 AltovaXMLXSLT2 = AltovaXML.XSLT2;
    AltovaXMLXSLT2.InputXMLFromText = "<?xml version='1.0' ?><doc>Hello
World</doc>";
    AltovaXMLXSLT2.XSLFileName = strExamplesFolder + "CopyInputXSLT2.xsl";
    String strResult = AltovaXMLXSLT2.ExecuteAndGetResultAsString();

// Show result
    MessageBox.Show("XSLT 2.0 engine answered: " + strResult);

```

### **Using .NET extensions**

```

// Specify folder (AltovaXMLExamples folder)
// Check if filepath is correct for you
    String strExamplesFolder = Environment.GetEnvironmentVariable
("ProgramFiles") + "\\Altova\\AltovaXML2011\\AltovaXMLExamples\\";

// Create a new AltovaXML instance and access its engines
    Altova.AltovaXML.Application AltovaXML = new Altova.AltovaXML.Application
();

// Use XSLT2 Engine from AltovaXML application
    Altova.AltovaXML.IXSLT2 AltovaXMLXSLT2 = AltovaXML.XSLT2;

// Enable .NET extensions
    AltovaXMLXSLT2.DotNetExtensionsEnabled = 1;

// Use XSLT containing .NET math extension for transformation
    AltovaXMLXSLT2.InputXMLFileName = strExamplesFolder + "simple.xml";
    AltovaXMLXSLT2.XSLFromText = "<xsl:stylesheet
xmlns:xsl='http://www.w3.org/1999/XSL/Transform'
    xmlns:math='clitype: System. Math' version='2.0'><xsl:output
omit-xml-declaration='yes' />
    <xsl:template match='/'><a><sqrtanswer><xsl:value-of select='

```

```

math: Sqrt(9) ' /></sqrtanswer></a>
  </xsl: template></xsl: stylesheet>";
  AltovaXMLXSLT2.Execute(strExamplesFolder + "Output.xml");

// Release ALL references to all components that were received.
System.Runtime.InteropServices.Marshal.ReleaseComObject(AltovaXMLXSLT2);
AltovaXMLXSLT2 = null;
System.Runtime.InteropServices.Marshal.ReleaseComObject(AltovaXML);
AltovaXML = null;

```

### Using the `LastErrorMessage` property

```

// Specify folder (AltovaXMLExamples folder)
// Check if filepath is correct for you
String strExamplesFolder = Environment.GetEnvironmentVariable
("ProgramFiles") + "\\Altova\\AltovaXML2011\\AltovaXMLExamples\\";

// Create a new AltovaXML instance and access its engines
Altova.AltovaXML.Application AltovaXML = new Altova.AltovaXML.Application
();

// Use XSLT2 Engine from AltovaXML application
Altova.AltovaXML.IXSLT2 AltovaXMLXSLT2 = AltovaXML.XSLT2;

// Enable/disable .NET extensions (true/false, 1/0)
AltovaXMLXSLT2.DotNetExtensionsEnabled = 0;

// Use XSLT containing .NET math extension for transformation
AltovaXMLXSLT2.InputXMLFileName = strExamplesFolder + "simple.xml";
AltovaXMLXSLT2.XSLFromText = "<xsl: stylesheet
xmlns:xsl=' http://www.w3.org/1999/XSL/Transform'
xmlns:math=' clitype: System.Math' version=' 2.0 '><xsl: output
omit-xml-declaration=' yes' />
<xsl: template match=' /' ><a><sqrtanswer><xsl: value-of select='
math: Sqrt(9) ' /></sqrtanswer></a>
</xsl: template></xsl: stylesheet>";
try
{
  AltovaXMLXSLT2.Execute(strExamplesFolder + "Output.xml");
}
catch (Exception)
{
  String strError = AltovaXMLXSLT2.LastErrorMessage;
  // Show result
  MessageBox.Show("XSLT 2.0 engine errors: " + strError);
}

// Release ALL references to all components that were received.
System.Runtime.InteropServices.Marshal.ReleaseComObject(AltovaXMLXSLT2);
AltovaXMLXSLT2 = null;
System.Runtime.InteropServices.Marshal.ReleaseComObject(AltovaXML);
AltovaXML = null;

```

## 2.4.5 Altova.AltovaXML.XQuery

### Description

The `Altova.AltovaXML.XQuery` object provides methods and properties to execute an XQuery 1.0 transformation using the Altova XQuery 1.0 Engine. Results can be saved to a file or returned as a string. The object also enables external XQuery variables to be passed to the XQuery document. The URLs of XQuery and XML files can be supplied as strings via the object's properties. Alternatively, the XML and XQuery documents can be constructed within the code as text strings.

**Note:** Where string inputs are to be interpreted as URLs, absolute paths should be used. If a relative path is used, a mechanism to resolve the relative path should be defined in the calling module.

### Methods

The following methods are available:

**Execute**(OutputFileName as String)

void **execute**(String outputFilename)

`Execute` executes an XQuery 1.0 transformation and saves the result to an output file, the name and location of which is provided as an input string to the `Execute` method. For example:

```
Execute("C:\OutputDoc.xml") .
```

**ExecuteAndGetResultAsString**( ) as String

`ExecuteAndGetResultAsString` executes an XQuery 1.0 transformation and returns the result as a UTF-16 text string. *See below for examples.*

**AddExternalVariable**(VarName as String, VarValue as String)

Takes a variable name and the value of this variable as input arguments. Each external variable and its value is to be specified in a separate call to the method. Variables must be declared in the XQuery document, optionally with a type declaration. Whatever the type declaration for the external variable in the XQuery document, the variable value submitted to the `AddExternalVariable` does not need any special delimiter, such as quotes. However, the lexical form must match that of the expected type (for example, a variable of type `xs:date` must have a value in the lexical form `2004-01-31`; a value in the lexical form `2004/Jan/01` will cause an error). Note that this also means that you cannot use an XQuery 1.0 function (for example, `current-date()`) as the value of an external variable (since the lexical form of the function as it is written will either not match the required data type (if the datatype is specified in the declaration of the external variable) or will be read as a string (if the datatype is not specified).) If multiple calls specify the same variable name the value set by the latest will be used.

**ClearExternalVariableList**( )

No argument should be provided. The `ClearExternalVariableList` clears the external variables list created with `AddExternalVariable` methods.

**Note:** Setting the optional XML document must always be done before query execution.

### Properties

*The following properties are defined:*

**XQueryFileName**

A string input that is read as a URL to locate the XQuery file to be executed. If both the `XQueryFileName` property and `XQueryFromText` property are specified, then the property that has been set later than the other (in the code sequence) is used.

**InputXMLFileName**

A string input that is read as a URL to locate the XML file that will be loaded into the query. XQuery navigation expressions are evaluated with reference to the document node of this XML document. If both the `InputXMLFileName` property and `InputXMLFromText` property are specified, then the property that has been set later than the other (in the code sequence) is used.

**XQueryFromText**

A string input that constructs an XQuery document. If both the `XQueryFileName` property and `XQueryFromText` property are specified, then the property that has been set later than the other (in the code sequence) is used.

**InputXMLFromText**

A string input that constructs an XML document. XQuery navigation expressions are evaluated with reference to the document node of this XML document. If both the `InputXMLFileName` property and `InputXMLFromText` property are specified, then the property that has been set later than the other (in the code sequence) is used.

**LastErrorMessage**

Returns the last error message.

**JavaExtensionsEnabled**

Enables Java extensions. You can specify whether Java extensions should be enabled or not by submitting `true` or `false` (case-insensitive) as a `Variant_Bool` argument.

**DotNetExtensionsEnabled**

Enables .NET extensions. You can specify whether .NET extensions should be enabled or not by submitting `true` or `false` (case-insensitive) as a `Variant_Bool` argument.

**Note:** If an XML document is set and is not needed for a new XQuery execution, then it should be cleared with an empty string assignment.

*The following serialization options are defined:*

**OutputMethod**

The required output method can be specified by submitting the required value as a string argument. Valid values are: `xml`, `xhtml`, `html`, and `text`. For example:

```
objAltovaXML.XQuery.OutputMethod = "xml". If the value is invalid, it is ignored. The default output method is xml.
```

**OutputOmitXMLDeclaration**

You can specify whether the XML declaration should be omitted or included in the output by submitting `true` or `false` (case-insensitive) as a Boolean argument. For example:

```
objAltovaXML.XQuery.OutputOmitXMLDeclaration = "FALSE". If the value is invalid, an error is raised. The default option is TRUE.
```

**OutputIndent**

You can specify whether the output should be indented or not by submitting `true` or `false` (case-insensitive) as a Boolean argument. For example:

```
objAltovaXML.XQuery.OutputIndent = "TRUE". If the value is invalid, an error is raised. The default option is False.
```

**OutputEncoding**

The required output encoding can be specified by submitting the encoding value as a string argument. For example: `objAltovaXML.XQuery.OutputEncoding = "UTF-8"`. If the value is

invalid, it is ignored. The default output encoding is UTF-8.

**Note:** For the serialization options, Raw Interface and Dispatch Interface usage differs. In the Raw Interface, if no argument is provided with these properties, then the current value of the property is returned. You would use something like: `put_OutputOption( VARIANT_BOOL bVal )` or `VARIANT_BOOL bVal = get_OutputOption()`, respectively, to set values and get values. In the Dispatch Interface, you can use `b = myXQuery.OutputOption` to get values and `myXQuery.OutputOption = b` to set values. For example, in the Dispatch Interface, `Sheet1.Cells(10, 2) = objAltovaXML.XQuery.OutputEncoding` would get the current output encoding.

### Example

The following C# code snippet shows how to validate an XML document and execute an XQuery. A [fuller example](#) is given at the end of the .NET Interface section.

To create the code snippet in a C# project, do the following:

1. In Microsoft Visual Studio, add a new project using **File | New | Project**.
2. Add a reference to the AltovaXML DLL by clicking **Project | Add Reference**. The Add Reference dialog pops up, displaying a list of installed .NET components. Select the AltovaXML component from the list to add it. (Note: If the AltovaXML component is not in the .NET tab list, it can be selected from the COM tab.)
3. Enter the example code snippet below in the project form. The code snippet below validates an XML file and executes an XQuery on the XML file. The files used in this code snippet are located in the `AltovaXMLExamples` folder of the AltovaXML application folder.
4. Compile the code and test it.

```
// Locate examples installed with AltovaXML
// REMARK: You might need to adapt this if you have a different major version
// of the product (2011 in this example).
String strExamplesFolder = Environment.GetEnvironmentVariable
("ProgramFiles") + "\\Altova\\AltovaXML2011\\AltovaXMLExamples\\";

// Create a new AltovaXML instance and access its engines
Altova.AltovaXML.Application AltovaXML = new Altova.AltovaXML.Application
();

// Validate input file simple.xml - it must be well-formed but not necessarily
// valid.
// The AltovaXML application will provide us with a validator object.
Altova.AltovaXML.XMLValidator AltovaXMLValidator = AltovaXML.XMLValidator;
AltovaXMLValidator.InputXMLFileName = strExamplesFolder + "simple.xml";
Boolean bIsWellFormed = AltovaXMLValidator.IsWellFormed();
Boolean bIsValid = AltovaXMLValidator.IsValid();

// Show result
MessageBox.Show("File " + strExamplesFolder + "simple.xml" + " is " +
    (bIsWellFormed ? "well-formed" : "not Well-formed") +
    " and " + (bIsValid ? "valid" : "invalid") + ".");

if (bIsWellFormed)
{
    // use XQuery Engine from AltovaXML application to transform simple.xml
    // with the help of CopyInput.xq
    Altova.AltovaXML.XQuery AltovaXMLXQuery = AltovaXML.XQuery;
    AltovaXMLXQuery.InputXMLFileName = strExamplesFolder + "simple.xml";
    AltovaXMLXQuery.XQueryFileName = strExamplesFolder + "CopyInput.xq";
    strResult = AltovaXMLXQuery.ExecuteAndGetResultAsString();
}
```



```
    // Show result  
    MessageBox.Show("XQuery engine answered: " + strResult);  
}
```

## 2.4.6 Example

The following C# code snippet shows how to validate an XML document, how to run XSLT 1.0 and XSLT 2.0 transformations, and how to execute an XQuery document. More code snippets are available in the preceding sections describing the individual engines: [XMLValidator](#); [XSLT1](#); [XSLT2](#); [XQuery](#).

To create the code snippet in a C# project, do the following:

1. In Microsoft Visual Studio, add a new project using **File | New | Project**.
2. Add a reference to the AltovaXML DLL by clicking **Project | Add Reference**. The Add Reference dialog pops up, displaying a list of installed .NET components. Select the AltovaXML component from the list to add it. (Note: If the AltovaXML component is not in the .NET tab list, it can be selected from the COM tab.)
3. Enter the example code snippet below in the project form. The code snippet below validates an XML file, runs XSLT transformations on the XML file using XSLT 1.0 and XSLT 2.0 stylesheets, and executes an XQuery document. The files used in this code snippet are located in the `AltovaXMLExamples` folder of the AltovaXML application folder.
4. Compile the code and test it.

```
// Locate examples installed with AltovaXML
// REMARK: You might need to adapt this if you have a different major version
// of the product (2011 in this example)
String strExamplesFolder = Environment.GetEnvironmentVariable
("ProgramFiles") + "\\Altova\\AltovaXML2011\\AltovaXMLExamples\\";

// Create a new AltovaXML instance and access its engines
Altova.AltovaXML.Application AltovaXML = new Altova.AltovaXML.Application
();

// Validate input file simple.xml - it must be well-formed but not necessarily
// valid.
// The AltovaXML application will provide us with a validator object.
Altova.AltovaXML.XMLValidator AltovaXMLValidator = AltovaXML.XMLValidator;
AltovaXMLValidator.InputXMLFileName = strExamplesFolder + "simple.xml";
Boolean bIsWellFormed = AltovaXMLValidator.IsWellFormed();
Boolean bIsValid = AltovaXMLValidator.IsValid();

// Show result
MessageBox.Show("File " + strExamplesFolder + "simple.xml" + " is " +
    (bIsWellFormed ? "well-formed" : "not Well-formed") +
    " and " + (bIsValid ? "valid" : "invalid") + ".");

// Release reference to XMLValidator component
System.Runtime.InteropServices.Marshal
.ReleaseComObject(AltovaXMLValidator);
AltovaXMLValidator = null;

if (bIsWellFormed)
{
    // Use XSLT1 Engine from the AltovaXML application to transform simple.
    // xml with the help of CopyInputXSLT1.xsl
    Altova.AltovaXML.IXSLT1 AltovaXMLXSLT1 = AltovaXML.XSLT1;
    AltovaXMLXSLT1.InputXMLFileName = strExamplesFolder + "simple.xml";
    AltovaXMLXSLT1.XSLFileName = strExamplesFolder + "CopyInputXSLT1.
xsl";

    String strResult = AltovaXMLXSLT1.ExecuteAndGetResultAsString();
    try
    {
        // Show result
        MessageBox.Show("XSLT 1.0 engine answered: " + strResult);
    }
}
```

```

    }
    catch (Exception)
    {
        String strError = AltovaXMLXSLT1.LastErrorMessage;
        // Show errors
        MessageBox.Show("XSLT 1.0 engine errors: " + strError);
    }

    // Release reference to XMLXSLT1 component
    System.Runtime.InteropServices.Marshal
.ReleaseComObject(AltovaXMLXSLT1);
    AltovaXMLXSLT1 = null;

    // use XSLT2 Engine from AltovaXML application to transform simple.xml
with the help of CopyInputXSLT2.xsl
    AltovaXMLXSLT2 = AltovaXML.XSLT2;
    AltovaXMLXSLT2.InputXMLFileName = strExamplesFolder + "simple.xml";
    AltovaXMLXSLT2.XSLFileName = strExamplesFolder + "CopyInputXSLT2.
xsl";

    strResult = AltovaXMLXSLT2.ExecuteAndGetResultAsString();
    try
    {
        // Show result
        MessageBox.Show("XSLT 2.0 engine answered: " + strResult);
    }
    catch (Exception)
    {
        String strError = AltovaXMLXSLT2.LastErrorMessage;
        // Show errors
        MessageBox.Show("XSLT 2.0 engine errors: " + strError);
    }

    // Release reference to XMLXSLT2 component
    System.Runtime.InteropServices.Marshal
.ReleaseComObject(AltovaXMLXSLT2);
    AltovaXMLXSLT2 = null;

    // use XQuery Engine from AltovaXML application to transform simple.xml
with the help of CopyInput.xq
    AltovaXMLXQuery = AltovaXML.XQuery;
    AltovaXMLXQuery.InputXMLFileName = strExamplesFolder + "simple.xml";
    AltovaXMLXQuery.XQueryFileName = strExamplesFolder + "CopyInput.xq";
    strResult = AltovaXMLXQuery.ExecuteAndGetResultAsString();
    try
    {
        // Show result
        MessageBox.Show("XQuery engine answered: " + strResult);
    }
    catch (Exception)
    {
        String strError = AltovaXMLXQuery.LastErrorMessage;
        // Show errors
        MessageBox.Show("XQuery engine errors: " + strError);
    }

    // Release reference to XMLXQuery component
    System.Runtime.InteropServices.Marshal
.ReleaseComObject(AltovaXMLXQuery);
    AltovaXMLXQuery = null;
}

// Release reference to AltovaXML component
System.Runtime.InteropServices.Marshal.ReleaseComObject(AltovaXML);
AltovaXML = null;

```

## 2.5 Explicitly Releasing AltovaXML COM-Server from C# and VB.NET

If references to the COM Server object are not released when the object gets descopeed, it is possible to explicitly release the AltovaXML COM references from within C# code using the `ReleaseComObject` methods as shown below.

Example:

```
private void button1_Click(object sender, EventArgs e)
{
    Altova.AltovaXML.ApplicationClass AltovaXML = new Altova.AltovaXML
.ApplicationClass();
    Altova.AltovaXML.IXSLT2 XSLT2 = AltovaXML.XSLT2;

    XSLT2.InputXMLFileName =
"C:\\Projects\\files\\XMLSpyExeFolder\\Examples\\OrgChart.xml";
    XSLT2.XSLFileName =
"C:\\Projects\\files\\XMLSpyExeFolder\\Examples\\OrgChart.xsl";
    XSLT2.Execute(
"C:\\Projects\\files\\XMLSpyExeFolder\\Examples\\OrgChart_out.html");

    // Release the XSLT2 component and then the AltovaXML component
    System.Runtime.InteropServices.Marshal.ReleaseComObject(XSLT2);
    XSLT2 = null;
    System.Runtime.InteropServices.Marshal.ReleaseComObject(AltovaXML
);
    AltovaXML = null;
}
```

- At the end of the method, the `AltovaXML.exe` **server** shuts down.
- If you do not call **all** of the `ReleaseComObject` methods, the exe servers will only be shut down with the shutdown of the C# application.

## 2.6 OOXML and ZIP Files

In order to enforce output to a ZIP file, including Open Office XML (OOXML) files such as `.docx`, one must specify the ZIP protocol in the file path. For example:

```
filename.zip| zip/filename.xxx  
filename.docx| zip/filename.xxx
```

In AltovaXML, ZIP file output can be specified with the following operations:

### COM interface and .NET interface

Output is generated using the `Execute` method. The argument of the method specifies the output file's name and location. For ZIP files, the ZIP protocol must be used, as in the following examples:

```
xslt2.Execute(c:\Mydocs\orgchart.zip| zip\main.xml)  
xslt2.Execute(c:\Mydocs\orgchart.docx| zip\main.out)  
xslt2.Execute(c:\Mydocs\orgchart.docx| zip\)
```

### Command line

When using the command line ensure that the output URI is enclosed in quotes. This is because the pipe character ( `|` ) would otherwise be interpreted by the command system. An example:

```
AltovaXML -in input.xml -xslt2 transform.xslt -out "c:\results.zipart.zip|  
zip\result.xml"
```

### The `xsl:result-document` element

In the case of the `xsl:result-document` element of XSLT 2.0, the ZIP protocol must be used on the output URI. In the case of OOXML documents, the ZIP protocol must be specified on the output URI of every `xsl:result-document` element involved in creating files for the OOXML document.

If the `xsl:result-document` elements specify relative output URIs, then specify the ZIP protocol for the main result, the URI of which is then used as the base URI to resolve the relative output URIs.



## **Chapter 3**

---

### **Engine Information**

### **3 Engine Information**

This section contains information about implementation-specific features of the Altova XML Validator, Altova XSLT 1.0 Engine, Altova XSLT 2.0 Engine, and Altova XQuery Engine.



### 3.1 Altova XML Validator

The Altova XML Validator implements and conforms to the rules of:

- [XML 1.0 \(Fourth Edition\)](#)
- [XML Namespaces \(1.0\)](#)
- [XML Schemas \(Structures\)](#)
- [XML Schema \(Datatypes\)](#)

## 3.2 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 whitespace-only text node that occurs between two elements within an element of mixed content.) This stripping may have an effect on the value returned by the `fn:position()`, `fn:last()`, and `fn:count()` functions.

For any node selection that selects text nodes also, boundary-whitespace-only text nodes would typically also be included in the selection. However, since the XML Infoset used by the Altova engines has boundary-whitespace-only text nodes stripped from it, these nodes are not present in the XML Infoset. As a result, the size of the selection and the numbering of nodes in the selection will be different than that for a selection which included these text nodes. The `fn:position()`, `fn:last()`, and `fn:count()` functions, therefore, could produce results that are different from those produced by some other processors.

A situation in which boundary-whitespace-only text nodes are evaluated as siblings of other elements arises most commonly when `xsl:apply-templates` is used to apply templates. When the `fn:position()`, `fn:last()`, and `fn:count()` functions are used in patterns with a name test (for example, `para[3]`, which is short for `para[position()=3]`), boundary-whitespace-only nodes are irrelevant since only the named elements (`para` in the above example) are selected. (Note, however, that boundary-whitespace-only nodes **are** relevant in patterns that use the wildcard, for example, `*[10]`.)

**Note:** If a boundary-whitespace-only text node is required in the output, then insert the required whitespace within one of the two adjoining child elements. For example, the XML fragment:

```
<para>This is <b>bold</b> <i>italic</i>.</para>
```

when processed with the XSLT template

```
<xsl:template match="para">
  <xsl:apply-templates/>
</xsl:template>
```

will produce:

```
This is bolditalic.
```

To get a space between **bold** and *italic* in the output, insert a space character within either the `<b>` or `<i>` elements in the XML source. For example:

```
<para>This is <b>bold</b> <i> italic</i>. </para> or  
<para>This is <b>bold</b> <i>italic</i>. </para> or  
<para>This is <b>bold</b><i> </i>italic</i>. </para>
```

When any of the `para` elements above is processed with the same XSLT template given above, it will produce:

```
This is bold italic.
```

### 3.3 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](#).

### 3.3.1 General Information

The Altova XSLT 2.0 Engine conforms to the World Wide Web Consortium's (W3C's) [XSLT 2.0 Recommendation](#) of 23 January 2007. Note the following general information about the engine.

#### Backwards Compatibility

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

In all other Altova products, the backwards-compatibility issue never arises. This is because these products automatically select the appropriate engine for the transformation. For example, consider that in XMLSpy you specify that a certain XML document be processed with an XSLT 1.0 stylesheet. When the transformation command is invoked, XMLSpy automatically selects the XSLT 1.0 Engine of XMLSpy to carry out the transformation.

**Note:** The stylesheet version is specified in the `version` attribute of the `stylesheet` or `transform` element of the stylesheet.

#### Namespaces

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

Namespace Name	Prefix	Namespace URI
XML Schema types	xs:	<a href="http://www.w3.org/2001/XMLSchema">http://www.w3.org/2001/XMLSchema</a>
XPath 2.0 functions	fn:	<a href="http://www.w3.org/2005/xpath-functions">http://www.w3.org/2005/xpath-functions</a>

Typically, these namespaces will be declared on the `xsl:stylesheet` or `xsl:transform` element, as shown in the following listing:

```
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:fn="http://www.w3.org/2005/xpath-functions"
  ...
</xsl:stylesheet>
```

The following points should be noted:

- The Altova XSLT 2.0 Engine uses the XPath 2.0 and XQuery 1.0 Functions namespace (listed in the table above) as its **default functions namespace**. So you can use XPath 2.0 and XSLT 2.0 functions in your stylesheet without any prefix. If you declare the XPath 2.0 Functions namespace in your stylesheet with a prefix, then you can additionally use the prefix assigned in the declaration.
- When using type constructors and types from the XML Schema namespace, the prefix used in the namespace declaration must be used when calling the type constructor (for example, `xs:date`).
- With the CRs of 23 January 2007, the `untypedAtomic` and `duration` datatypes ( `dayTimeDuration` and `yearMonthDuration`), which were formerly in the XPath Datatypes namespace (typically prefixed `xd:`) have been moved to the XML Schema namespace.

- Some XPath 2.0 functions have the same name as XML Schema datatypes. For example, for the XPath functions `fn:string` and `fn:boolean` there exist XML Schema datatypes with the same local names: `xs:string` and `xs:boolean`. So if you were to use the XPath expression `string(' Hello' )`, the expression evaluates as `fn:string(' Hello' )` —not as `xs:string(' Hello' )`.

### Schema-awareness

The Altova XSLT 2.0 Engine is schema-aware.

### Whitespace in XML document

By default, the Altova XSLT 2.0 Engine strips all boundary whitespace from boundary-whitespace-only nodes in the source XML document. The removal of this whitespace affects the values that the `fn:position()`, `fn:last()`, `fn:count()`, and `fn:deep-equal()` functions return. For more details, see [Whitespace-only Nodes in XML Document](#) in the XPath 2.0 and XQuery 1.0 Functions section.

**Note:** If a boundary-whitespace-only text node is required in the output, then insert the required whitespace within one of the two adjoining child elements. For example, the XML fragment:

```
<para>This is <b>bold</b> <i>italic</i>.</para>
```

when processed with the XSLT template

```
<xsl:template match="para">
  <xsl:apply-templates/>
</xsl:template>
```

will produce:

```
This is bolditalic.
```

To get a space between `bold` and `italic` in the output, insert a space character within either the `<b>` or `<i>` elements in the XML source. For example:

```
<para>This is <b>bold</b> <i> italic</i>.</para> or
<para>This is <b>bold<#x20;</b> <i>italic</i>.</para> or
<para>This is <b>bold</b><i>&#x20;italic</i>.</para>
```

When such an XML fragment is processed with the same XSLT template given above, it will produce:

```
This is bold italic.
```

### XSLT 2.0 elements and functions

Limitations and implementation-specific behavior of XSLT 2.0 elements and functions are listed in the section [XSLT 2.0 Elements and Functions](#).

### XPath 2.0 functions

Implementation-specific behavior of XPath 2.0 functions is listed in the section [XPath 2.0 and XQuery 1.0 Functions](#).

### 3.3.2 XSLT 2.0 Elements and Functions

#### Limitations

The `xsl:preserve-space` and `xsl:strip-space` elements are not supported.

#### Implementation-specific behavior

Given below is a description of how the Altova XSLT 2.0 Engine handles implementation-specific aspects of the behavior of certain XSLT 2.0 functions.

#### `xsl:result-document`

Additionally supported encodings are: `x-base16tobinary` and `x-base64tobinary`.

#### `function-available`

The function tests for the availability of in-scope functions (XSLT 2.0, XPath 2.0, and extension functions).

#### `unparsed-text`

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

#### `unparsed-text-available`

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

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

### 3.4 XQuery 1.0 Engine: Implementation Information

The Altova XQuery 1.0 Engine is built into Altova's XMLSpy and MapForce XML products. It is also available in the free AltovaXML package. This section provides information about implementation-defined aspects of behavior.

#### Standards conformance

The Altova XQuery 1.0 Engine conforms to the World Wide Web Consortium's (W3C's) [XQuery 1.0 Recommendation](#) of 23 January 2007. The XQuery standard gives implementations discretion about how to implement many features. Given below is a list explaining how the Altova XQuery 1.0 Engine implements these features.

#### Schema awareness

The Altova XQuery 1.0 Engine is **schema-aware**.

#### Encoding

The UTF-8 and UTF-16 character encodings are supported.

#### Namespaces

The following namespace URIs and their associated bindings are pre-defined.

Namespace Name	Prefix	Namespace URI
XML Schema types	xs:	<a href="http://www.w3.org/2001/XMLSchema">http://www.w3.org/2001/XMLSchema</a>
Schema instance	xsi:	<a href="http://www.w3.org/2001/XMLSchema-instance">http://www.w3.org/2001/XMLSchema-instance</a>
Built-in functions	fn:	<a href="http://www.w3.org/2005/xpath-functions">http://www.w3.org/2005/xpath-functions</a>
Local functions	local:	<a href="http://www.w3.org/2005/xquery-local-functions">http://www.w3.org/2005/xquery-local-functions</a>

The following points should be noted:

- The Altova XQuery 1.0 Engine recognizes the prefixes listed above as being bound to the corresponding namespaces.
- Since the built-in functions namespace listed above is the default functions namespace in XQuery, the `fn:` prefix does not need to be used when built-in functions are invoked (for example, `string("Hello")` will call the `fn:string` function). However, the prefix `fn:` can be used to call a built-in function without having to declare the namespace in the query prolog (for example: `fn:string("Hello")`).
- You can change the default functions namespace by declaring the `default function namespace` expression in the query prolog.
- When using types from the XML Schema namespace, the prefix `xs:` may be used without having to explicitly declare the namespaces and bind these prefixes to them in the query prolog. (Example: `xs:date` and `xs:yearMonthDuration`.) If you wish to use some other prefix for the XML Schema namespace, this must be explicitly declared in the query prolog. (Example: `declare namespace alt = "http://www.w3.org/2001/XMLSchema"; alt:date("2004-10-04")`.)
- Note that the `untypedAtomic`, `dayTimeDuration`, and `yearMonthDuration` datatypes have been moved, with the CRs of 23 January 2007, from the XPath Datatypes namespace to the XML Schema namespace, so: `xs:yearMonthDuration`.

If namespaces for functions, type constructors, node tests, etc are wrongly assigned, an error is



reported. Note, however, that some functions have the same name as schema datatypes, e.g. `fn:string` and `fn:boolean`. (Both `xs:string` and `xs:boolean` are defined.) The namespace prefix determines whether the function or type constructor is used.

### XML source document and validation

XML documents used in executing an XQuery document with the Altova XQuery 1.0 Engine must be well-formed. However, they do not need to be valid according to an XML Schema. If the file is not valid, the invalid file is loaded without schema information. If the XML file is associated with an external schema and is valid according to it, then post-schema validation information is generated for the XML data and will be used for query evaluation.

### Static and dynamic type checking

The static analysis phase checks aspects of the query such as syntax, whether external references (e.g. for modules) exist, whether invoked functions and variables are defined, and so on. No type checking is done in the static analysis phase. If an error is detected in the static analysis phase, it is reported and the execution is stopped.

Dynamic type checking is carried out at run-time, when the query is actually executed. If a type is incompatible with the requirement of an operation, an error is reported. For example, the expression `xs:string("1") + 1` returns an error because the addition operation cannot be carried out on an operand of type `xs:string`.

### Library Modules

Library modules store functions and variables so they can be reused. The Altova XQuery 1.0 Engine supports modules that are stored in **a single external XQuery file**. Such a module file must contain a `module` declaration in its prolog, which associates a target namespace. Here is an example module:

```
module namespace libns="urn:module-library";
declare variable $libns:company := "Altova";
declare function libns:webaddress() { "http://www.altova.com" };
```

All functions and variables declared in the module belong to the namespace associated with the module. The module is used by importing it into an XQuery file with the `import module` statement in the query prolog. The `import module` statement only imports functions and variables declared directly in the library module file. As follows:

```
import module namespace modlib = "urn:module-library" at
    "modulefilename.xq";
if ($modlib:company = "Altova")
then modlib:webaddress()
else error("No match found.")
```

### External functions

External functions are not supported, i.e. in those expressions using the `external` keyword, as in:

```
declare function hoo($param as xs:integer) as xs:string external;
```

### Collations

The default collation is the Unicode codepoint collation. No other collation is currently supported. Comparisons, including the `fn:max` function, are based on this collation.

**Character normalization**

No character normalization form is supported.

**Precision of numeric types**

- The `xs:integer` datatype is arbitrary-precision, i.e. it can represent any number of digits.
- The `xs:decimal` datatype has a limit of 20 digits after the decimal point.
- The `xs:float` and `xs:double` datatypes have limited-precision of 15 digits.

**XQuery Instructions Support**

The `Pragma` instruction is not supported. If encountered, it is ignored and the fallback expression is evaluated.

**XQuery Functions Support**

For information about implementation-specific behavior of XQuery 1.0 functions, see the section, [XPath 2.0 and XQuery 1.0 Functions](#).

### 3.5 XPath 2.0 and XQuery 1.0 Functions

XPath 2.0 and XQuery 1.0 functions are evaluated by:

- the **Altova XPath 2.0 Engine**, which (i) is a component of the Altova XSLT 2.0 Engine, and (ii) is used in the XPath Evaluator of Altova's XMLSpy product to evaluate XPath expressions with respect to the XML document that is active in the XMLSpy interface.
- the **Altova XQuery 1.0 Engine**.

This section describes how XPath 2.0 and XQuery 1.0 functions are handled by the Altova XPath 2.0 Engine and Altova XQuery 1.0 Engine. Only those functions are listed, for which the behavior is implementation-specific, or where the behavior of an individual function is different in any of the three environments in which these functions are used (that is, in XSLT 2.0, in XQuery 1.0, and in the XPath Evaluator of XMLSpy). Note that this section does not describe how to use these functions. For more information about the usage of functions, see the World Wide Web Consortium's (W3C's) [XQuery 1.0 and XPath 2.0 Functions and Operators Recommendation](#) of 23 January 2007.

### 3.5.1 General Information

#### Standards conformance

- The Altova XPath 2.0 Engine implements the World Wide Web Consortium's (W3C's) [XPath 2.0 Recommendation](#) of 23 January 2007. The Altova XQuery 1.0 Engine implements the World Wide Web Consortium's (W3C's) [XQuery 1.0 Recommendation](#) of 23 January 2007. The XPath 2.0 and XQuery 1.0 functions support in these two engines is compliant with the [XQuery 1.0 and XPath 2.0 Functions and Operators Recommendation](#) of 23 January 2007.
- The Altova XPath 2.0 Engine conforms to the rules of [XML 1.0 \(Fourth Edition\)](#) and [XML Namespaces \(1.0\)](#).

#### Default functions namespace

The default functions namespace has been set to comply with that specified in the standard. Functions can therefore be called without a prefix.

#### Boundary-whitespace-only nodes in source XML document

The XML data (and, consequently, the XML Infoset) that is passed to the Altova XPath 2.0 Engine and Altova XQuery 1.0 Engine is stripped of boundary-whitespace-only text nodes. (A boundary-whitespace-only text node is a child whitespace-only text node that occurs between two elements within an element of mixed content.) This stripping has an effect on the value returned by the `fn:position()`, `fn:last()`, `fn:count()`, and `fn:deep-equal()` functions.

For any node selection that selects text nodes also, boundary-whitespace-only text nodes would typically also be included in the selection. However, since the XML Infoset used by the Altova engines has boundary-whitespace-only text nodes stripped from it, these nodes are not present in the XML Infoset. As a result, the size of the selection and the numbering of nodes in the selection will be different than that for a selection which included these text nodes. The `fn:position()`, `fn:last()`, `fn:count()`, and `fn:deep-equal()` functions, therefore, could produce results that are different from those produced by some other processors.

A situation in which boundary-whitespace-only text nodes are evaluated as siblings of other elements arises most commonly when `xsl:apply-templates` is used to apply templates. When the `fn:position()`, `fn:last()`, and `fn:count()` functions are used in patterns with a name test (for example, `para[3]`, which is short for `para[position()=3]`), boundary-whitespace-only nodes are irrelevant since only the named elements (`para` in the above example) are selected. (Note, however, that boundary-whitespace-only nodes **are** relevant in patterns that use the wildcard, for example, `*[10]`.)

#### Numeric notation

On output, when an `xs:double` is converted to a string, scientific notation (for example, `1.0E12`) is used when the absolute value is less than 0.000001 or greater than 1,000,000. Otherwise decimal or integer notation is used.

#### Precision of `xs:decimal`

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

#### Implicit timezone

When two `date`, `time`, or `dateTime` values need to be compared, the timezone of the values being compared need to be known. When the timezone is not explicitly given in such a value, the implicit timezone is used. The implicit timezone is taken from the system clock, and its value can be checked with the `fn:implicit-timezone()` function.

**Collations**

Only the Unicode codepoint collation is supported. No other collations can be used. String comparisons, including for the `fn:max` and `fn:min` functions, are based on this collation.

**Namespace axis**

The namespace axis is deprecated in XPath 2.0. Use of the namespace axis is, however, supported. To access namespace information with XPath 2.0 mechanisms, use the `fn:in-scope-prefixes()`, `fn:namespace-uri()` and `fn:namespace-uri-for-prefix()` functions.

**Static typing extensions**

The optional static type checking feature is not supported.

### 3.5.2 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
<code>base-uri</code>	<ul style="list-style-type: none"> <li>• If external entities are used in the source XML document and if a node in the external entity is specified as the input node argument of the <code>base-uri()</code> function, it is still the base URI of the including XML document that is used—not the base URI of the external entity.</li> <li>• The base URI of a node in the XML document can be modified using the <code>xml:base</code> attribute.</li> </ul>
<code>collection</code>	<ul style="list-style-type: none"> <li>• The argument is a relative URI that is resolved against the current base URI.</li> <li>• If the resolved URI identifies an XML file, then this XML file is treated as a catalog which references a collection of files. This file must have the form: <pre>&lt;collection&gt;   &lt;doc href="uri-1" /&gt;   &lt;doc href="uri-2" /&gt;   &lt;doc href="uri-3" /&gt; &lt;/collection&gt;</pre> <p>The files referenced by the <code>href</code> attributes are loaded, and their document nodes are returned as a sequence.</p> </li> <li>• If the resolved URI does not identify an XML file with the catalog structure described above, then the argument string (in which wildcards such as <code>?</code> and <code>*</code> are allowed) is used as a search string. XML files with names that match the search expression are loaded, and their document nodes are returned as a sequence. See examples below.</li> <li>• XSLT example: The expression <code>collection("c:\MyDocs\*.xml")//Title</code> returns a sequence of all <code>DocTitle</code> elements in the <code>.xml</code> files in the <code>MyDocs</code> folder.</li> <li>• XQuery example: The expression <code>{for \$i in collection(c:\MyDocs\*.xml) return element doc{base-uri(\$i)}}</code> returns the base URIs of all the <code>.xml</code> files in the <code>MyDocs</code> folder, each URI being within a <code>doc</code> element.</li> <li>• The default collection is empty.</li> </ul>

Function Name	Notes
<code>count</code>	<ul style="list-style-type: none"> <li>• See note on whitespace in the <a href="#">General Information</a> section.</li> </ul>

current-date, current-dateTime, current-time	<ul style="list-style-type: none"> <li>• The current date and time is taken from the system clock.</li> <li>• The timezone is taken from the implicit timezone provided by the evaluation context; the implicit timezone is taken from the system clock.</li> <li>• The timezone is always specified in the result.</li> </ul>
deep-equal	<ul style="list-style-type: none"> <li>• See note on whitespace in the <a href="#">General Information</a> section.</li> </ul>
doc	<ul style="list-style-type: none"> <li>• 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.</li> </ul>
id	<ul style="list-style-type: none"> <li>• 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.</li> </ul>
in-scope-prefixes	<ul style="list-style-type: none"> <li>• 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.</li> </ul>
last	<ul style="list-style-type: none"> <li>• See note on whitespace in the <a href="#">General Information</a> section.</li> </ul>
lower-case	<ul style="list-style-type: none"> <li>• The Unicode character set is supported.</li> </ul>
normalize-unicode	<ul style="list-style-type: none"> <li>• The normalization forms NFC, NFD, NFKC, and NFKD are supported.</li> </ul>

Function Name	Notes
position	<ul style="list-style-type: none"> <li>• See note on whitespace in the <a href="#">General Information</a> section.</li> </ul>
resolve-uri	<ul style="list-style-type: none"> <li>• 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.</li> <li>• The relative URI (the first argument) is appended after the last "/" in the path notation of the base URI notation.</li> <li>• 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).</li> </ul>

<code>static-base-uri</code>	<ul style="list-style-type: none"><li>• 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.</li><li>• When using XPath Evaluator in the XMLSpy IDE, the base URI from the static context is the URI of the active XML document.</li></ul>
<code>upper-case</code>	<ul style="list-style-type: none"><li>• The Unicode character set is supported.</li></ul>



## 3.6 Extensions

There are several ready-made functions in programming languages such as Java and C# that are not available as XPath 2.0 / XQuery 1.0 functions or as XSLT 2.0 functions. A good example of such functions are the math functions available in Java, such as `sin()` and `cos()`. If these functions were available to the designers of XSLT stylesheets and XQuery queries, it would increase the application area of stylesheets and queries and greatly simplify the tasks of stylesheet creators.

Altova Engines (XSLT 1.0, XSLT 2.0, and XQuery 1.0), which are used in a number of Altova products, support the use of extension functions in Java and .NET. The Altova XSLT Engines additionally support MSXSL scripts for XSLT 1.0 and 2.0 and Altova's own extension functions.

You should note that extension functions are always called from XPath expressions. This section describes how to use extension functions and MSXSL scripts in your XSLT stylesheets and XQuery queries. These descriptions are organized into the following sections:

- [Java Extension Functions](#)
- [.NET Extension Functions](#)
- [MSXSL Scripts for XSLT](#)
- [Altova Extension Functions](#)

The two main issues considered in the descriptions are: (i) how functions in the respective libraries are called; and (ii) what rules are followed for converting arguments in a function call to the required input format of the function, and what rules are followed for the return conversion (function result to XSLT/XQuery data object).

### Requirements

For extension functions support, a Java Runtime Environment (for access to Java functions) and .NET Framework 2.0 (minimum, for access to .NET functions) must be installed on the machine running the XSLT transformation or XQuery execution, or must be accessible for the transformations.

### 3.6.1 Java Extension Functions

A Java extension function can be used within an XPath or XQuery expression to invoke a Java constructor or call a Java method (static or instance).

A field in a Java class is considered to be a method without any argument. A field can be static or instance. How to access fields is described in the respective sub-sections, static and instance.

This section is organized into the following sub-sections:

- [Java: Constructors](#)
- [Java: Static Methods and Static Fields](#)
- [Java: Instance Methods and Instance Fields](#)
- [Datatypes: XSLT/XQuery to Java](#)
- [Datatypes: Java to XSLT/XQuery](#)

#### Form of the extension function

The extension function in the XPath/XQuery expression must have the form `prefix:fname()`.

- The `prefix:` part identifies the extension function as a Java function. It does so by associating the extension function with an in-scope namespace declaration, the URI of which must begin with `java:` (*see below for examples*). The namespace declaration should identify a Java class, for example: `xmlns:myns="java:java.lang.Math"`. However, it could also simply be: `xmlns:myns="java"` (without a colon), with the identification of the Java class being left to the `fname()` part of the extension function.
- The `fname()` part identifies the Java method being called, and supplies the arguments for the method (*see below for examples*). However, if the namespace URI identified by the `prefix:` part does not identify a Java class (*see preceding point*), then the Java class should be identified in the `fname()` part, before the class and separated from the class by a period (*see the second XSLT example below*).

**Note:** The class being called must be on the classpath of the machine.

#### XSLT example

Here are two examples of how a static method can be called. In the first example, the class name (`java.lang.Math`) is included in the namespace URI and, therefore, must not be in the `fname()` part. In the second example, the `prefix:` part supplies the prefix `java:` while the `fname()` part identifies the class as well as the method.

```
<xsl:value-of xmlns:jMath="java:java.lang.Math"
  select="jMath:cos(3.14)" />

<xsl:value-of xmlns:jmath="java"
  select="jmath:java.lang.Math.cos(3.14)" />
```

The method named in the extension function (`cos()` in the example above) must match the name of a public static method in the named Java class (`java.lang.Math` in the example above).

#### XQuery example

Here is an XQuery example similar to the XSLT example above:

```
<cosine xmlns:jMath="java:java.lang.Math">
  { jMath:cos(3.14) }
</cosine>
```

### User-defined Java classes

If you have created your own Java classes, methods in these classes are called differently according to: (i) whether the classes are accessed via a JAR file or a class file, and (ii) whether these files (JAR or class) are located in the current directory (the same directory as the XSLT or XQuery document) or not. How to locate these files is described in the sections [User-Defined Class Files](#) and [User-Defined Jar Files](#). Note that paths to class files not in the current directory and to all JAR files must be specified.

### User-Defined Class Files

If access is via a class file, then there are two possibilities:

- The class file is in a package. The XSLT or XQuery file is in the same folder as the Java package.
- The class file is not packaged. The XSLT or XQuery file is in the same folder as the class file.
- The class file is in a package. The XSLT or XQuery file is at some random location.
- The class file is not packaged. The XSLT or XQuery file is at some random location.

Consider the case where the class file is not packaged and is in the same folder as the XSLT or XQuery document. In this case, since all classes in the folder are found, the file location does not need to be specified. The syntax to identify a class is:

```
java: classname
```

*where*

`java:` indicates that a user-defined Java function is being called; (Java classes in the current directory will be loaded by default)

`classname` is the name of the required method's class

The class is identified in a namespace URI, and the namespace is used to prefix a method call.

### Class file packaged, XSLT/XQuery file in same folder as Java package

The example below calls the `getVehicleType()` method of the `Car` class of the `com.altova.extfunc` package. The `com.altova.extfunc` package is in the folder `JavaProject`. The XSLT file is also in the folder `JavaProject`.

```
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:fn="http://www.w3.org/2005/xpath-functions"
  xmlns:car="java:com.altova.extfunc.Car" >
<xsl:output exclude-result-prefixes="fn car xsl fo xs"/>

<xsl:template match="/">
  <a>
    <xsl:value-of select="car:getVehicleType()" />
  </a>
</xsl:template>
</xsl:stylesheet>
```

**Class file not packaged, XSLT/XQuery file in same folder as class file**

The example below calls the `getVehicleType()` method of the `Car` class of the `com.altova.extfunc` package. The `Car` class file is in the following folder location: `JavaProject/com/altova/extfunc`. The XSLT file is also in the folder `JavaProject/com/altova/extfunc`.

```
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:fn="http://www.w3.org/2005/xpath-functions"
  xmlns:car="java:Car" >
<xsl:output exclude-result-prefixes="fn car xsl fo xs"/>

<xsl:template match="/">
  <a>
    <xsl:value-of select="car:getVehicleType()" />
  </a>
</xsl:template>

</xsl:stylesheet>
```

**Class file packaged, XSLT/XQuery file at any location**

The example below calls the `getCarColor()` method of the `Car` class of the `com.altova.extfunc` package. The `com.altova.extfunc` package is in the folder `JavaProject`. The XSLT file is at any location. In this case, the location of the package must be specified within the URI as a query string. The syntax is:

```
java:classname[?path=uri-of-package]
```

**where**

`java:` indicates that a user-defined Java function is being called  
`uri-of-package` is the URI of the Java package  
`classname` is the name of the required method's class

The class is identified in a namespace URI, and the namespace is used to prefix a method call. The example below shows how to access a class file that is located in another directory than the current directory.

```
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:fn="http://www.w3.org/2005/xpath-functions"
  xmlns:car="
java:com.altova.extfunc.Car?path=file:///C:/JavaProject/" >

<xsl:output exclude-result-prefixes="fn car xsl xs"/>

<xsl:template match="/">
  <xsl:variable name="myCar" select="car:new('red') " />
  <a><xsl:value-of select="car:getCarColor($myCar)" /></a>
</xsl:template>

</xsl:stylesheet>
```

**Class file not packaged, XSLT/XQuery file at any location**

The example below calls the `getCarColor()` method of the `Car` class of the

`com.altova.extfunc` package. The `com.altova.extfunc` package is in the folder `JavaProject`. The XSLT file is at any location. The location of the class file is specified within the namespace URI as a query string. The syntax is:

```
java:classname[?path=uri-of-classfile]
```

#### *where*

`java:` indicates that a user-defined Java function is being called  
`uri-of-classfile` is the URI of the folder containing the class file  
`classname` is the name of the required method's class

The class is identified in a namespace URI, and the namespace is used to prefix a method call. The example below shows how to access a class file that is located in another directory than the current directory.

```
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:fn="http://www.w3.org/2005/xpath-functions"
  xmlns:car="
java:Car?path=file:///C:/JavaProject/com/altova/extfunc/" >

  <xsl:output exclude-result-prefixes="fn car xsl xs"/>

  <xsl:template match="/">
    <xsl:variable name="myCar" select="car:new('red')"/>
    <a><xsl:value-of select="car:getCarColor($myCar)"/></a>
  </xsl:template>

</xsl:stylesheet>
```

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

### User-Defined Jar Files

If access is via a JAR file, the URI of the JAR file must be specified using the following syntax:

```
xmlns:classNS="java:classname?path=jar:uri-of-jarfile!/"
```

The method is then called by using the prefix of the namespace URI that identifies the `class: classNS: method()`

#### *In the above:*

`java:` indicates that a Java function is being called  
`classname` is the name of the user-defined class  
`?` is the separator between the classname and the path  
`path=jar:` indicates that a path to a JAR file is being given  
`uri-of-jarfile` is the URI of the jar file  
`!/` is the end delimiter of the path  
`classNS: method()` is the call to the method

Alternatively, the classname can be given with the method call. Here are two examples of the syntax:

```

xmlns: ns1="java: docx. layout. pages?path=jar: file: ///c: /projects/docs/docx. jar! /
"
    ns1: main()

xmlns: ns2="java?path=jar: file: ///c: /projects/docs/docx. jar! /"
ns2: docx. layout. pages. main()

```

Here is a complete XSLT example that uses a JAR file to call a Java extension function:

```

<xsl: stylesheet version="2.0"
    xmlns: xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns: xs="http://www.w3.org/2001/XMLSchema"
    xmlns: fn="http://www.w3.org/2005/xpath-functions"
    xmlns: car="java?path=jar: file: ///C: /test/Carl. jar! /" >
<xsl: output exclude-result-prefixes="fn car xsl xs"/>

<xsl: template match="/">
    <xsl: variable name="myCar" select="car: Carl. new(' red' )" />
    <a><xsl: value-of select="car: Carl. getCarColor( $myCar) "/></a>
</xsl: template>

<xsl: template match="car"/>

</xsl: stylesheet>

```

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

### Java: Constructors

An extension function can be used to call a Java constructor. All constructors are called with the pseudo-function `new()`.

If the result of a Java constructor call can be [implicitly converted to XPath/XQuery datatypes](#), then the Java extension function will return a sequence that is an XPath/XQuery datatype. If the result of a Java constructor call cannot be converted to a suitable XPath/XQuery datatype, then the constructor creates a wrapped Java object with a type that is the name of the class returning that Java object. For example, if a constructor for the class `java.util.Date` is called ( `java.util.Date.new()` ), then an object having a type `java.util.Date` is returned. The lexical format of the returned object may not match the lexical format of an XPath datatype and the value would therefore need to be converted to the lexical format of the required XPath datatype and then to the required XPath datatype.

There are two things that can be done with a Java object created by a constructor:

- It can be assigned to a variable:
 

```
<xsl: variable name="currentdate" select="date: new() " xmlns: date="
java: java. util. Date" />
```
- It can be passed to an extension function (see [Instance Method and Instance Fields](#)):
 

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

### Java: Static Methods and Static Fields

A static method is called directly by its Java name and by supplying the arguments for the method. Static fields (methods that take no arguments), such as the constant-value fields `E` and `PI`, are accessed without specifying any argument.

### XSLT examples

Here are some examples of how static methods and fields can be called:

```
<xsl:value-of xmlns:jMath="java:java.lang.Math"
  select="jMath:cos(3.14)" />

<xsl:value-of xmlns:jMath="java:java.lang.Math"
  select="jMath:cos(jMath:PI())" />

<xsl:value-of xmlns:jMath="java:java.lang.Math"
  select="jMath:E() * jMath:cos(3.14)" />
```

Notice that the extension functions above have the form `prefix:fname()`. The prefix in all three cases is `jMath:`, which is associated with the namespace URI `java:java.lang.Math`. (The namespace URI must begin with `java:`. In the examples above it is extended to contain the class name (`java.lang.Math`.) The `fname()` part of the extension functions must match the name of a public class (e.g. `java.lang.Math`) followed by the name of a public static method with its argument/s (such as `cos(3.14)`) or a public static field (such as `PI()`).

In the examples above, the class name has been included in the namespace URI. If it were not contained in the namespace URI, then it would have to be included in the `fname()` part of the extension function. For example:

```
<xsl:value-of xmlns:java="java:"
  select="java:java.lang.Math.cos(3.14)" />
```

### XQuery example

A similar example in XQuery would be:

```
<cosine xmlns:jMath="java:java.lang.Math">
  {jMath:cos(3.14)}
</cosine>
```

### Java: Instance Methods and Instance Fields

An instance method has a Java object passed to it as the first argument of the method call. Such a Java object typically would be created by using an extension function (for example a constructor call) or a stylesheet parameter/variable. An XSLT example of this kind would be:

```
<xsl:stylesheet version="1.0" exclude-result-prefixes="date"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:date="java:java.util.Date"
  xmlns:jlang="java:java.lang">
  <xsl:param name="CurrentDate" select="date:new()" />
  <xsl:template match="/">
    <enrollment institution-id="Altova School"
      date="{date:toString($CurrentDate)}"
      type="{jlang:Object.toString(jlang:Object.getClass(date:new()
    ))}">
    </enrollment>
  </xsl:template>
</xsl:stylesheet>
```

In the example above, the value of the node `enrollment/@type` is created as follows:

1. An object is created with a constructor for the class `java.util.Date` (with the `date:new()` constructor).
2. This Java object is passed as the argument of the `jlang.Object.getClass` method.
3. The object obtained by the `getClass` method is passed as the argument to the `jlang.Object.toString` method.

The result (the value of `@type`) will be a string having the value: `java.util.Date`.

An instance field is theoretically different from an instance method in that it is not a Java object per se that is passed as an argument to the instance field. Instead, a parameter or variable is passed as the argument. However, the parameter/variable may itself contain the value returned by a Java object. For example, the parameter `CurrentDate` takes the value returned by a constructor for the class `java.util.Date`. This value is then passed as an argument to the instance method `date: toString` in order to supply the value of `/enrollment/@date`.

### Datatypes: XPath/XQuery to Java

When a Java function is called from within an XPath/XQuery expression, the datatype of the function's arguments is important in determining which of multiple Java classes having the same name is called.

In Java, the following rules are followed:

- If there is more than one Java method with the same name, but each has a different number of arguments than the other/s, then the Java method that best matches the number of arguments in the function call is selected.
- The XPath/XQuery string, number, and boolean datatypes (*see list below*) are implicitly converted to a corresponding Java datatype. If the supplied XPath/XQuery type can be converted to more than one Java type (for example, `xs:integer`), then that Java type is selected which is declared for the selected method. For example, if the Java method being called is `fx(decimal)` and the supplied XPath/XQuery datatype is `xs:integer`, then `xs:integer` will be converted to Java's `decimal` datatype.

The table below lists the implicit conversions of XPath/XQuery string, number, and boolean types to Java datatypes.

<code>xs:string</code>	<code>java.lang.String</code>
<code>xs:boolean</code>	<code>boolean (primitive)</code> , <code>java.lang.Boolean</code>
<code>xs:integer</code>	<code>int</code> , <code>long</code> , <code>short</code> , <code>byte</code> , <code>float</code> , <code>double</code> , and the wrapper classes of these, such as <code>java.lang.Integer</code>
<code>xs:float</code>	<code>float (primitive)</code> , <code>java.lang.Float</code> , <code>double (primitive)</code>
<code>xs:double</code>	<code>double (primitive)</code> , <code>java.lang.Double</code>
<code>xs:decimal</code>	<code>float (primitive)</code> , <code>java.lang.Float</code> , <code>double (primitive)</code> , <code>java.lang.Double</code>

Subtypes of the XML Schema datatypes listed above (and which are used in XPath and XQuery) will also be converted to the Java type/s corresponding to that subtype's ancestor type.

In some cases, it might not be possible to select the correct Java method based on the supplied information. For example, consider the following case.

- The supplied argument is an `xs:untypedAtomic` value of 10 and it is intended for the method `mymethod(float)`.
- However, there is another method in the class which takes an argument of another datatype: `mymethod(double)`.



- Since the method names are the same and the supplied type (`xs: untypedAtomic`) could be converted correctly to either `float` or `double`, it is possible that `xs: untypedAtomic` is converted to `double` instead of `float`.
- Consequently the method selected will not be the required method and might not produce the expected result. To work around this, you can create a user-defined method with a different name and use this method.

Types that are not covered in the list above (for example `xs: date`) will not be converted and will generate an error. However, note that in some cases, it might be possible to create the required Java type by using a Java constructor.

### Datatypes: Java to XPath/XQuery

When a Java method returns a value, the datatype of the value is a string, numeric or boolean type, then it is converted to the corresponding XPath/XQuery type. For example, Java's `java.lang.Boolean` and `boolean` datatypes are converted to `xsd:boolean`.

One-dimensional arrays returned by functions are expanded to a sequence. Multi-dimensional arrays will not be converted, and should therefore be wrapped.

When a wrapped Java object or a datatype other than string, numeric or boolean is returned, you can ensure conversion to the required XPath/XQuery type by first using a Java method (e.g. `toString`) to convert the Java object to a string. In XPath/XQuery, the string can be modified to fit the lexical representation of the required type and then converted to the required type (for example, by using the `cast as` expression).

### 3.6.2 .NET Extension Functions

If you are working on the .NET platform, you can use extension functions written in any of the .NET languages (for example, C#). A .NET extension function can be used within an XPath or XQuery expression to invoke a constructor, property, or method (static or instance) within a .NET class.

A property of a .NET class is called using the syntax `get_PropertyName()`.

This section is organized into the following sub-sections:

- [.NET: Constructors](#)
- [.NET: Static Methods and Static Fields](#)
- [.NET: Instance Methods and Instance Fields](#)
- [Datatypes: XSLT/XQuery to .NET](#)
- [Datatypes: .NET to XSLT/XQuery](#)

#### Form of the extension function

The extension function in the XPath/XQuery expression must have the form `prefix:fname()`.

- The `prefix:` part is associated with a URI that identifies the .NET class being addressed.
- The `fname()` part identifies the constructor, property, or method (static or instance) within the .NET class, and supplies any argument/s, if required.
- The URI must begin with `clitype:` (which identifies the function as being a .NET extension function).
- The `prefix:fname()` form of the extension function can be used with system classes and with classes in a loaded assembly. However, if a class needs to be loaded, additional parameters containing the required information will have to be supplied.

#### Parameters

To load an assembly, the following parameters are used:

<code>asm</code>	The name of the assembly to be loaded.
<code>ver</code>	The version number (maximum of four integers separated by periods).
<code>sn</code>	The key token of the assembly's strong name (16 hex digits).
<code>from</code>	A URI that gives the location of the assembly (DLL) to be loaded. If the URI is relative, it is relative to the XSLT or XQuery document. If this parameter is present, any other parameter is ignored.
<code>partialname</code>	The partial name of the assembly. It is supplied to <code>Assembly.LoadWith.PartialName()</code> , which will attempt to load the assembly. If <code>partialname</code> is present, any other parameter is ignored.
<code>loc</code>	The locale, for example, <code>en-US</code> . The default is <code>neutral</code> .

If the assembly is to be loaded from a DLL, use the `from` parameter and omit the `sn` parameter. If the assembly is to be loaded from the Global Assembly Cache (GAC), use the `sn` parameter and omit the `from` parameter.

A question mark must be inserted before the first parameter, and parameters must be separated by a semi-colon. The parameter name gives its value with an equals sign (see *example below*).

### Examples of namespace declarations

An example of a namespace declaration in XSLT that identifies the system class

`System.Environment`:

```
xmlns:myns="clitype:System.Environment"
```

An example of a namespace declaration in XSLT that identifies the class to be loaded as

`Trade.Forward.Scrip`:

```
xmlns:myns="clitype:Trade.Forward.Scrip?asm=forward;version=10.6.2.1"
```

An example of a namespace declaration in XQuery that identifies the system class

`MyManagedDLL.testClass`:. Two cases are distinguished:

1. When the assembly is loaded from the GAC:

```
declare namespace cs="clitype:MyManagedDLL.testClass?asm=MyManagedDLL;
ver=1.2.3.4;loc=neutral;sn=b9f091b72dccfba8";
```

2. When the assembly is loaded from the DLL (complete and partial references below):

```
declare namespace
cs="clitype:MyManagedDLL.testClass?from=file:///C:/Altova
Projects/extFunctions/MyManagedDLL.dll;

declare namespace
cs="clitype:MyManagedDLL.testClass?from=MyManagedDLL.dll;
```

### XSLT example

Here is a complete XSLT example that calls functions in system class `System.Math`:

```
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:fn="http://www.w3.org/2005/xpath-functions">
  <xsl:output method="xml" omit-xml-declaration="yes" />
  <xsl:template match="/">
    <math xmlns:math="clitype:System.Math">
      <sqrt><xsl:value-of select="math:Sqrt(9)"/></sqrt>
      <pi><xsl:value-of select="math:PI()"/></pi>
      <e><xsl:value-of select="math:E()"/></e>
      <pow><xsl:value-of select="math:Pow(math:PI(), math:E()"/></pow>
    </math>
  </xsl:template>
</xsl:stylesheet>
```

The namespace declaration on the element `math` associates the prefix `math:` with the URI `clitype:System.Math`. The `clitype:` beginning of the URI indicates that what follows identifies either a system class or a loaded class. The `math:` prefix in the XPath expressions associates the extension functions with the URI (and, by extension, the class) `System.Math`. The extension functions identify methods in the class `System.Math` and supply arguments where required.

### XQuery example

Here is an XQuery example fragment similar to the XSLT example above:

```
<math xmlns:math="clitype:System.Math">
  { math:Sqrt(9) }
</math>
```

As with the XSLT example above, the namespace declaration identifies the .NET class, in this

case a system class. The XQuery expression identifies the method to be called and supplies the argument.

### .NET: Constructors

An extension function can be used to call a .NET constructor. All constructors are called with the pseudo-function `new()`. If there is more than one constructor for a class, then the constructor that most closely matches the number of arguments supplied is selected. If no constructor is deemed to match the supplied argument/s, then a 'No constructor found' error is returned.

#### Constructors that return XPath/XQuery datatypes

If the result of a .NET constructor call can be [implicitly converted to XPath/XQuery datatypes](#), then the .NET extension function will return a sequence that is an XPath/XQuery datatype.

#### Constructors that return .NET objects

If the result of a .NET constructor call cannot be converted to a suitable XPath/XQuery datatype, then the constructor creates a wrapped .NET object with a type that is the name of the class returning that object. For example, if a constructor for the class `System.DateTime` is called (with `System.DateTime.new()`), then an object having a type `System.DateTime` is returned.

The lexical format of the returned object may not match the lexical format of a required XPath datatype. In such cases, the returned value would need to be: (i) converted to the lexical format of the required XPath datatype; and (ii) cast to the required XPath datatype.

There are three things that can be done with a .NET object created by a constructor:

- It can be used within a variable:  

```
<xsl:variable name="currentdate" select="date:new( 2008, 4, 29) "
xmlns:date="clitype:System.DateTime" />
```
- It can be passed to an extension function (see [Instance Method and Instance Fields](#)):  

```
<xsl:value-of select="date:ToString( date:new( 2008, 4, 29) )" xmlns:date
="clitype:System.DateTime" />
```
- It can be converted to a string, number, or boolean:
- ```
<xsl:value-of select="xs:integer( data:get_Month( date:new( 2008, 4, 29) ))
" xmlns:date="clitype:System.DateTime" />
```

### .NET: Static Methods and Static Fields

A static method is called directly by its name and by supplying the arguments for the method. The name used in the call must exactly match a public static method in the class specified. If the method name and the number of arguments that were given in the function call matches more than one method in a class, then the types of the supplied arguments are evaluated for the best match. If a match cannot be found unambiguously, an error is reported.

**Note:** A field in a .NET class is considered to be a method without any argument. A property is called using the syntax `get_PropertyName()`.

#### Examples

An XSLT example showing a call to a method with one argument (`System.Math.Sin(arg)`):

```
<xsl:value-of select="math:Sin( 30) " xmlns:math="clitype:System.Math"/>
```

An XSLT example showing a call to a field (considered a method with no argument) (`System.Double.MaxValue()`):

```
<xsl:value-of select="double:MaxValue()" xmlns:double="
clitype:System.Double"/>
```

An XSLT example showing a call to a property (syntax is `get_PropertyName()`) (`System.String()`):

```
<xsl:value-of select="string:get_Length(' my string') " xmlns:string="
clitype:System.String"/>
```

An XQuery example showing a call to a method with one argument (`System.Math.Sin(arg)`):

```
<sin xmlns:math="clitype:System.Math">
  { math:Sin(30) }
</sin>
```

### .NET: Instance Methods and Instance Fields

An instance method has a .NET object passed to it as the first argument of the method call. This .NET object typically would be created by using an extension function (for example a constructor call) or a stylesheet parameter/variable. An XSLT example of this kind would be:

```
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:fn="http://www.w3.org/2005/xpath-functions">
  <xsl:output method="xml" omit-xml-declaration="yes"/>
  <xsl:template match="/">
    <xsl:variable name="releasedate"
      select="date:new(2008, 4, 29)"
      xmlns:date="clitype:System.DateTime"/>
    <doc>
      <date>
        <xsl:value-of select="date:ToString(date:new(2008, 4, 29))"
          xmlns:date="clitype:System.DateTime"/>
      </date>
      <date>
        <xsl:value-of select="date:ToString($releasedate)"
          xmlns:date="clitype:System.DateTime"/>
      </date>
    </doc>
  </xsl:template>
</xsl:stylesheet>
```

In the example above, a `System.DateTime` constructor (`new(2008, 4, 29)`) is used to create a .NET object of type `System.DateTime`. This object is created twice, once as the value of the variable `releasedate`, a second time as the first and only argument of the `System.DateTime.ToString()` method. The instance method `System.DateTime.ToString()` is called twice, both times with the `System.DateTime` constructor (`new(2008, 4, 29)`) as its first and only argument. In one of these instances, the variable `releasedate` is used to get the .NET object.

### Instance methods and instance fields

The difference between an instance method and an instance field is theoretical. In an instance method, a .NET object is directly passed as an argument; in an instance field, a parameter or variable is passed instead—though the parameter or variable may itself contain a .NET object. For example, in the example above, the variable `releasedate` contains a .NET object, and it is this variable that is passed as the argument of `ToString()` in the second `date` element constructor. Therefore, the `ToString()` instance in the first `date` element is an instance method while the second is considered to be an instance field. The result produced in both instances, however, is the same.

### Datatypes: XPath/XQuery to .NET

When a .NET extension function is used within an XPath/XQuery expression, the datatypes of the function's arguments are important for determining which one of multiple .NET methods having the same name is called.

In .NET, the following rules are followed:

- If there is more than one method with the same name in a class, then the methods available for selection are reduced to those that have the same number of arguments as the function call.
- The XPath/XQuery string, number, and boolean datatypes (see *list below*) are implicitly converted to a corresponding .NET datatype. If the supplied XPath/XQuery type can be converted to more than one .NET type (for example, `xs:integer`), then that .NET type is selected which is declared for the selected method. For example, if the .NET method being called is `fx(double)` and the supplied XPath/XQuery datatype is `xs:integer`, then `xs:integer` will be converted to .NET's `double` datatype.

The table below lists the implicit conversions of XPath/XQuery string, number, and boolean types to .NET datatypes.

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

Subtypes of the XML Schema datatypes listed above (and which are used in XPath and XQuery) will also be converted to the .NET type/s corresponding to that subtype's ancestor type.

In some cases, it might not be possible to select the correct .NET method based on the supplied information. For example, consider the following case.

- The supplied argument is an `xs:untypedAtomic` value of 10 and it is intended for the method `mymethod(float)`.
- However, there is another method in the class which takes an argument of another datatype: `mymethod(double)`.
- Since the method names are the same and the supplied type (`xs:untypedAtomic`) could be converted correctly to either `float` or `double`, it is possible that `xs:untypedAtomic` is converted to `double` instead of `float`.
- Consequently the method selected will not be the required method and might not produce the expected result. To work around this, you can create a user-defined

method with a different name and use this method.

Types that are not covered in the list above (for example `xs:date`) will not be converted and will generate an error.

### **Datatypes: .NET to XPath/XQuery**

When a .NET method returns a value and the datatype of the value is a string, numeric or boolean type, then it is converted to the corresponding XPath/XQuery type. For example, .NET's `decimal` datatype is converted to `xs:decimal`.

When a .NET object or a datatype other than string, numeric or boolean is returned, you can ensure conversion to the required XPath/XQuery type by first using a .NET method (for example `System.DateTime.ToString()`) to convert the .NET object to a string. In XPath/XQuery, the string can be modified to fit the lexical representation of the required type and then converted to the required type (for example, by using the `cast as` expression).

### 3.6.3 MSXSL Scripts for XSLT

The `<msxsl:script>` element contains user-defined functions and variables that can be called from within XPath expressions in the XSLT stylesheet. The `<msxsl:script>` is a top-level element, that is, it must be a child element of `<xsl:stylesheet>` or `<xsl:transform>`.

The `<msxsl:script>` element must be in the namespace `urn:schemas-microsoft-com:xslt` (see *example below*).

#### Scripting language and namespace

The scripting language used within the block is specified in the `<msxsl:script>` element's `language` attribute and the namespace to be used for function calls from XPath expressions is identified with the `implements-prefix` attribute (see *below*).

```
<msxsl:script language="scripting-language" implements-prefix="user-namespace-prefix">
    function-1 or variable-1
    ...
    function-n or variable-n
</msxsl:script>
```

The `<msxsl:script>` element interacts with the Windows Scripting Runtime, so only languages that are installed on your machine may be used within the `<msxsl:script>` element. **The .NET Framework 2.0 platform or higher must be installed for MSXSL scripts to be used.** Consequently, the .NET scripting languages can be used within the `<msxsl:script>` element.

The `language` attribute accepts the same values as the `language` attribute on the HTML `<script>` element. If the `language` attribute is not specified, then Microsoft JScript is assumed as the default.

The `implements-prefix` attribute takes a value that is a prefix of a declared in-scope namespace. This namespace typically will be a user namespace that has been reserved for a function library. All functions and variables defined within the `<msxsl:script>` element will be in the namespace identified by the prefix specified in the `implements-prefix` attribute. When a function is called from within an XPath expression, the fully qualified function name must be in the same namespace as the function definition.

#### Example

Here is an example of a complete XSLT stylesheet that uses a function defined within a `<msxsl:script>` element.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:fn="http://www.w3.org/2005/xpath-functions"
  xmlns:msxsl="urn:schemas-microsoft-com:xslt"
  xmlns:user="http://mycompany.com/mynamespace">
  <msxsl:script language="VBScript" implements-prefix="user">
    <![CDATA[
      ' Input: A currency value: the wholesale price
      ' Returns: The retail price: the input value plus 20% margin,
      ' rounded to the nearest cent
      dim a as integer = 13
      Function AddMargin(WholesalePrice) as integer
```



```

        AddMargin = WholesalePrice * 1.2 + a
    End Function
]]>
</msxsl:script>

<xsl:template match="/">
  <html>
    <body>
      <p>
        <b>Total Retail Price =
          $<xsl:value-of select="user: AddMargin( 50)"/>
        </b>
        <br/>
        <b>Total Wholesale Price =
          $<xsl:value-of select="50"/>
        </b>
      </p>
    </body>
  </html>
</xsl:template>
</xsl:stylesheet>

```

### Datatypes

The values of parameters passed into and out of the script block are limited to XPath datatypes. This restriction does not apply to data passed among functions and variables within the script block.

### Assemblies

An assembly can be imported into the script by using the `msxsl: assembly` element. The assembly is identified via a name or a URI. The assembly is imported when the stylesheet is compiled. Here is a simple representation of how the `msxsl: assembly` element is to be used.

```

<msxsl:script>
  <msxsl:assembly name="myAssembly.assemblyName" />
  <msxsl:assembly href="pathToAssembly" />

  ...

</msxsl:script>

```

The assembly name can be a full name, such as:

```
"system.Math, Version=3.1.4500.1 Culture=neutral
PublicKeyToken=a46b3f648229c514"
```

or a short name, such as "myAssembly.Draw".

### Namespaces

Namespaces can be declared with the `msxsl: using` element. This enables assembly classes to be written in the script without their namespaces, thus saving you some tedious typing. Here is how the `msxsl: using` element is used so as to declare namespaces.

```

<msxsl:script>
  <msxsl:using namespace="myAssemblyNS.NamespaceName" />

  ...

</msxsl:script>

```

The value of the `namespace` attribute is the name of the namespace.



### 3.6.4 Altova Extension Functions

Altova extension functions are in the namespace `http://www.altova.com/xslt-extensions` and are indicated in this section with the prefix `altova:`, which is assumed to be bound to the namespace given above.

The following extension functions are supported in the current version of your Altova product in the manner described below. However, note that in future versions of your product, support for one or more of these functions might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

#### General functions

- [altova: evaluate\(\)](#)
- [altova: distinct-nodes\(\)](#)
- [altova: encode-for-rtf\(\)](#)
- [altova: xbrl-labels\(\)](#)
- [altova: xbrl-footnotes\(\)](#)
- [altova: generate-auto-number\(\)](#)
- [altova: reset-auto-number\(\)](#)
- [altova: get-temp-folder\(\)](#)

#### Chart functions (Enterprise and Reporting Editions only)

Altova extension functions for charts are supported only in the Enterprise and Reporting Editions of Altova products and enable charts to be generated from XML data. The chart functions have been organized into two groups:

- [Functions to generate and save charts](#)
- [Functions to create charts](#)

A third section gives a listing of the [chart data XML structure](#), from which charts can be generated. Finally, an [example XSLT document](#) shows how chart functions can be used to generate charts from XML data.

#### General Functions

The following extension functions are supported in the current version of your Altova product in the manner described below. However, note that in future versions of your product, support for one or more of these functions might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

- [altova: evaluate\(\)](#)
- [altova: distinct-nodes\(\)](#)
- [altova: encode-for-rtf\(\)](#)
- [altova: xbrl-labels\(\)](#)
- [altova: xbrl-footnotes\(\)](#)
- [altova: generate-auto-number\(\)](#)
- [altova: reset-auto-number\(\)](#)
- [altova: get-temp-folder\(\)](#)

**altova:evaluate()**

The `altova:evaluate()` function takes an XPath expression, passed as a string, as its mandatory argument. It returns the output of the evaluated expression.

```
altova:evaluate( XPathExp as xs:string )
```

For example:

```
altova:evaluate( ' //Name[ 1] ' )
```

In the example above, note that the expression `//Name[ 1]` is passed as a string by enclosing it in single quotes. The `altova:evaluate` function returns the contents of the first `Name` element in the document.

The `altova:evaluate` function can take additional (optional) arguments. These arguments are, respectively, the values of variables with the names `p1`, `p2`, `p3... pN` that can be used in the XPath expression.

```
altova:evaluate( XPathExp as xs:string [, p1value ... pNvalue] )
```

where

- the variable names must be of the form `pX`, `X` being an integer
- the sequence of the function's arguments, from the second argument onwards corresponds to the sequence of variables named `p1` to `pN`. So the second argument will be the value of the variable `p1`, the third argument that of the variable `p2`, and so on.
- The variable values must be of type `item*`

For example:

```
<xs1:variable name="xpath" select="$p3, $p2, $p1" />
<xs1:value-of select="altova:evaluate( $xpath, 10, 20, 'hi' )" />
Outputs "hi 20 10"
```

In the above listing, notice the following:

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

The following examples further illustrate usage:

```
<xs1:variable name="xpath" select="$p1" />
<xs1:value-of select="altova:evaluate( $xpath, //Name[1] )" />
Outputs value of the first Name element.

<xs1:variable name="xpath" select="$p1" />
<xs1:value-of select="altova:evaluate( $xpath, '//Name[1]' )" />
Outputs "//Name[ 1] "
```

The `altova:evaluate()` extension function is useful in situations where an XPath expression in the XSLT stylesheet contains one or more parts that must be evaluated dynamically. For example, consider a situation in which a user enters his request for the sorting criterion and this criterion is stored in the attribute `UserReq/@sortkey`. In the stylesheet, you could then have the

expression :

```
<xsl:sort select="altova:evaluate(.. /UserReq/@sortkey)" order="ascending"/>
```

The `altova:evaluate()` function reads the `sortkey` attribute of the `UserReq` child element of the parent of the context node. Say the value of the `sortkey` attribute is `Price`, then `Price` is returned by the `altova:evaluate()` function and becomes the value of the `select` attribute:

```
<xsl:sort select="Price" order="ascending"/>
```

If this `sort` instruction occurs within the context of an element called `Order`, then the `Order` elements will be sorted according to the values of their `Price` children. Alternatively, if the value of `@sortkey` were, say, `Date`, then the `Order` elements would be sorted according to the values of their `Date` children. So the sort criterion for `Order` is selected from the `sortkey` attribute at runtime. This could not have been achieved with an expression like:

```
<xsl:sort select=".. /UserReq/@sortkey" order="ascending"/>
```

In the case shown above, the sort criterion would be the `sortkey` attribute itself, not `Price` or `Date` (or any other current content of `sortkey`).

Variables can be used in the `altova:evaluate()` extension function as shown in the examples below:

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

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

#### **altova:distinct-nodes()**

The `altova:distinct-nodes()` function takes a set of one or more nodes as its input and returns the same set minus nodes with duplicate values. The comparison is done using the XPath/XQuery function `fn:deep-equal`.

```
altova:distinct-nodes( $arg as node()* ) as node()*
```

#### **altova:encode-for-rtf()**

The `altova:encode-for-rtf()` function converts the input string into code for RTF.

```
altova:encode-for-rtf( $inputstr as xs:string?,
  $preserveallwhitespace as xs:boolean,
  $preservenewlines as xs:boolean) as xs:string
```

Whitespace and new lines will be preserved according to the boolean value specified for their

respective parameters.

**altova: xbrl-labels()**

The `altova: xbrl-labels()` function takes two input arguments: a node name and the taxonomy file location containing the node. The function returns the XBRL labels associated with the input node.

```
altova: xbrl-labels( $name as xs:QName, $file as xs:string ) as node()*
```

**altova: xbrl-footnotes()**

The `altova: xbrl-footnotes()` function takes a node as its input argument and returns the set of XBRL footnote nodes referenced by the input node.

```
altova: xbrl-footnotes( $arg as node() ) as node()*
```

**altova: generate-auto-number(id as xs:string, start-with as xs:integer, increment as xs:integer, reset-on-change as xs:string)**

Generates a series of numbers having the specified ID. The start integer and the increment is specified.

**altova: reset-auto-number(id as xs:string)**

This function resets the auto-numbering of the auto-numbering series specified with the ID argument. The series is reset to the start integer of the series (see `altova: generate-auto-number` above).

**altova: get-temp-folder as xs:string**

Gets the temporary folder.

### Barcode Functions

The Altova XSLT Engines use third-party Java libraries to create barcodes. Given below are the classes and the public methods used. The classes are packaged in

`AltovaBarcodeExtension.jar`, which is located in the folder `C:\Program Files\Altova\CommonYYYY\jar`.

The Java libraries used are in sub-folders of the folder `C:\Program Files\Altova\CommonYYYY\jar`:

- `barcode4j\barcode4j.jar` (Website: <http://barcode4j.sourceforge.net/>)
- `zxing\core.jar` (Website: <http://code.google.com/p/zxing/>)

The license files are also located in the respective folders.

**The `com.altova.extensions.barcode` package**

The package, `com.altova.extensions.barcode`, is used to generate most of the barcode types.

The following classes are used:

```
public class BarcodeWrapper
    static BarcodeWrapper newInstance( String name, String msg, int dpi, int
orientation, BarcodePropertyWrapper[] arrProperties )
    double getHeightPlusQuiet()
    double getWidthPlusQuiet()
    org.w3c.dom.Document generateBarcodeSVG()
    byte[] generateBarcodePNG()
    String generateBarcodePngAsHexString()
```

public class **BarcodePropertyWrapper** *Used to store the barcode properties that will be dynamically set later*

```
BarcodePropertyWrapper( String methodName, String propertyValue )
BarcodePropertyWrapper( String methodName, Integer propertyValue
)
BarcodePropertyWrapper( String methodName, Double propertyValue )
BarcodePropertyWrapper( String methodName, Boolean propertyValue )
BarcodePropertyWrapper( String methodName, Character propertyValue )
String getMethodName()
Object getPropertyValue()
```

public class **AltovaBarcodeClassResolver** *Registers the class com.altova.extensions.barcode.proxy.zxing.QRCodeBean for the qrcode bean, additionally to the classes registered by the org.krysalis.barcode4j.DefaultBarcodeClassResolver.*

### The com.altova.extensions.barcode.proxy.zxing package

The package, com.altova.extensions.barcode.proxy.zxing, is used to generate the QRCode barcode type.

The following classes are used:

```
class QRCodeBean
    • Extends org.krysalis.barcode4j.impl.AbstractBarcodeBean
    • Creates an AbstractBarcodeBean interface for com.google.zxing.qrcode.encoder

void generateBarcode(CanvasProvider canvasImp, String msg)
void setQRErrorCorrectionLevel(QRCodeErrorCorrectionLevel level)
BarcodeDimension calcDimensions(String msg)
double getVerticalQuietZone()
double getBarWidth()
```

```
class QRCodeErrorCorrectionLevel Error correction level for the QRCode
    static QRCodeErrorCorrectionLevel byName(String name)
    "L" = ~7% correction
    "M" = ~15% correction
    "H" = ~25% correction
    "Q" = ~30% correction
```

### XSLT example

Given below is an XSLT example showing how barcode functions are used in an XSLT stylesheet.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
```

```

xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:fn="http://www.w3.org/2005/xpath-functions"
xmlns:altova="http://www.altova.com"
xmlns:altovaext="http://www.altova.com/xslt-extensions"
xmlns:altovaext-barcode="java:com.altova.extensions.barcode.BarcodeWrapper"

xmlns:altovaext-barcode-property="
java:com.altova.extensions.barcode.BarcodePropertyWrapper">
<xsl:output method="html" encoding="UTF-8" indent="yes"/>
<xsl:template match="/">
  <html>
    <head><title/></head>
    <body>
      
    </body>
  </html>
<xsl:result-document
  href="{altovaext:get-temp-folder()}barcode.png"
  method="text" encoding="base64tobinary" >
  <xsl:variable name="barcodeObject"
    select="
altovaext-barcode:newInstance('Code39',string('some
value'),
  96,0,(altovaext-barcode-property:new('setWidth',
25.4 div 96 * 2) ) )"/>
  <xsl:value-of select="
xs:base64Binary(xs:hexBinary(string(altovaext-barcode:generateBarcodePngAsHexS
tring($barcodeObject) ) )"/>
  </xsl:result-document>
</xsl:template>
</xsl:stylesheet>

```

## Chart Functions

The chart functions listed below enable you to create, generate, and save charts as images. They are supported in the current version of your Altova product in the manner described below. However, note that in future versions of your product, support for one or more of these functions might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

The chart functions are organized into two groups:

- [Functions for generating and saving charts](#)
- [Functions for creating charts](#)

**Note:** Chart functions are supported only in the **Enterprise and Reporting Editions** of Altova products.

### Functions for generating and saving charts

These functions take the chart object (obtained with the chart creation functions) and either generate an image or save an image to file

**altova:generate-chart-image** (\$chart, \$width, \$height, \$encoding) as atomic

where

- \$chart is the chart extension item obtained with the `altova:create-chart` function



- `$width` and `$height` must be specified with a length unit
- `$encoding` may be `binarytobase64` or `binarytobase16`

The function returns the chart image in the specified encoding.

```
altova: generate-chart-image($chart, $width, $height, $encoding, $imagetype)  
as atomic
```

where

- `$chart` is the chart extension item obtained with the `altova: create-chart` function
- `$width` and `$height` must be specified with a length unit
- `$encoding` may be `base64Binary` or `hexBinary`
- `$imagetype` may be one of the following image formats: `png`, `gif`, `bmp`, `jpg`, `jpeg`

The function returns the chart image in the specified encoding and image format.

```
altova: save-chart-image($chart, $filename, $width, $height) as empty()
```

where

- `$chart` is the chart extension item obtained with the `altova: create-chart` function
- `$filename` is the path to and name of the file to which the chart image is to be saved
- `$width` and `$height` must be specified with a length unit

The function saves the chart image to the file specified in `$filename`.

```
altova: save-chart-image($chart, $filename, $width, $height, $imagetype) as  
empty()
```

where

- `$chart` is the chart extension item obtained with the `altova: create-chart` function
- `$filename` is the path to and name of the file to which the chart image is to be saved
- `$width` and `$height` must be specified with a length unit
- `$imagetype` may be one of the following image formats: `png`, `gif`, `bmp`, `jpg`, `jpeg`

The function saves the chart image to the file specified in `$filename` in the image format specified.

### Functions for creating charts

The following functions are used to create charts.

```
altova: create-chart($chart-config, $chart-data-series*) as chart extension  
item
```

where

- `$chart-config` is the `chart-config` extension item obtained with the `altova: create-chart-config` function or via the `altova: create-chart-config-from-xml` function
- `$chart-data-series` is the `chart-data-series` extension item obtained with the

altova: create-chart-data-series function or  
altova: create-chart-data-series-from-rows function

The function returns a chart extension item, which is created from the data supplied via the arguments.

**altova: create-chart-config**(\$type-name, \$title) as chart-config extension item

where

- \$type-name specifies the type of chart to be created: Pie, Pie3d, BarChart, BarChart3d, BarChart3dGrouped, LineChart, ValueLineChart, RoundGauge, BarGauge
- \$title is the name of the chart

The function returns a chart-config extension item containing the configuration information of the chart.

**altova: create-chart-config-from-xml**(\$xml-struct) as chart-config extension item

where

- \$xml-struct is the XML structure containing the configuration information of the chart

The function returns a chart-config extension item containing the configuration information of the chart. This information is supplied in an [XML data fragment](#).

**altova: create-chart-data-series**(\$series-name?, \$x-values\*, \$y-values\*) as chart-data-series extension item

where

- \$series-name specifies the name of the series
- \$x-values gives the list of X-Axis values
- \$y-values gives the list of Y-Axis values

The function returns a chart-data-series extension item containing the data for building the chart: that is, the names of the series and the Axes data.

**altova: create-chart-data-row**(x, y1, y2, y3, ...) as chart-data-x-Ny-row extension item

where

- x is the value of the X-Axis column of the chart data row
- yN are the values of the Y-Axis columns

The function returns a chart-data-x-Ny-row extension item, which contains the data for the X-Axis column and Y-Axis columns of a single series.

**altova: create-chart-data-series-from-rows**(\$series-names as xs:string\*, \$row\*) as chart-data-series extension item

where

- `$series-name` is the name of the series to be created
- `$row` is the `chart-data-x-Ny-row` extension item that is to be created as a series

The function returns a `chart-data-series` extension item, which contains the data for the X-Axis and Y-Axes of the series.

**altova: create-chart-layer**(`$chart-config`, `$chart-data-series*`) as `chart-layer` extension item

where

- `$chart-config` is the `chart-config` extension item obtained with the `altova: create-chart-config` function or or via the `altova: create-chart-config-from-xml` function
- `$chart-data-series` is the `chart-data-series` extension item obtained with the `altova: create-chart-data-series` function or `altova: create-chart-data-series-from-rows` function

The function returns a `chart-layer` extension item, which contains chart-layer data.

**altova: create-multi-layer-chart**(`$chart-config`, `$chart-data-series*`, `$chart-layer*`)

where

- `$chart-config` is the `chart-config` extension item obtained with the `altova: create-chart-config` function or or via the `altova: create-chart-config-from-xml` function
- `$chart-data-series` is the `chart-data-series` extension item obtained with the `altova: create-chart-data-series` function or `altova: create-chart-data-series-from-rows` function
- `$chart-layer` is the `chart-layer` extension item obtained with the `altova: create-chart-layer` function

The function returns a `multi-layer-chart` item.

**altova: create-multi-layer-chart**(`$chart-config`, `$chart-data-series*`, `$chart-layer*`, `xs:boolean $mergecategoryvalues`)

where

- `$chart-config` is the `chart-config` extension item obtained with the `altova: create-chart-config` function or or via the `altova: create-chart-config-from-xml` function
- `$chart-data-series` is the `chart-data-series` extension item obtained with the `altova: create-chart-data-series` function or `altova: create-chart-data-series-from-rows` function
- `$chart-layer` is the `chart-layer` extension item obtained with the `altova: create-chart-layer` function

The function returns a `multi-layer-chart` item.

## Chart Data XML Structure

Given below is the XML structure of chart data, how it might appear for the [Altova extension functions for charts](#). This affects the appearance of the specific chart. Not all elements are used for all chart kinds, e.g. the <Pie> element is ignored for bar charts.

**Note:** Chart functions are supported only in the **Enterprise and Reporting Editions** of Altova products.

```
<chart-config>
  <General
    SettingsVersion="1"  must be provided
    ChartKind="BarChart"  Pie, Pie3d, BarChart, StackedBarChart, BarChart3d,
    BarChart3dGrouped, LineChart, ValueLineChart, AreaChart, StackedAreaChart, RoundGauge,
    BarGauge, CandleStick
    BKColor="#ffffff"  Color
    BKColorGradientEnd="#ffffff"  Color. In case of a gradient, BKColor and
    BKColorGradientEnd define the gradient's colors
    BKMode="#ffffff"  Solid, HorzGradient, VertGradient
    BKFile="Path+Filename"  String. If file exists, its content is drawn over the
    background.
    BKFileMode="Stretch"  Stretch, ZoomToFit, Center, Tile
    ShowBorder="1"  Bool
    PlotBorderColor="#000000"  Color
    PlotBKColor="#ffffff"  Color
    Title=""  String
    ShowLegend="1"  Bool
    OutsideMargin="3.%"  PercentOrPixel
    TitleToPlotMargin="3.%"  PercentOrPixel
    LegendToPlotMargin="3.%"  PercentOrPixel
    Orientation="vert"  Enumeration: possible values are: vert, horz
  >
  <TitleFont
    Color="#000000"  Color
    Name="Tahoma"  String
    Bold="1"  Bool
    Italic="0"  Bool
    Underline="0"  Bool
    MinFontHeight="10.pt"  FontSize (only pt values)
    Size="8.%"  FontSize />
  <LegendFont
    Color="#000000"
    Name="Tahoma"
    Bold="0"
    Italic="0"
    Underline="0"
    MinFontHeight="10.pt"
    Size="3.5%" />
  <AxisLabelFont
    Color="#000000"
    Name="Tahoma"
    Bold="1"
    Italic="0"
    Underline="0"
    MinFontHeight="10.pt"
    Size="5.%" />
</General>
```

```

<Line
  ConnectionShapeSize="1.%" PercentOrPixel
  DrawFilledConnectionShapes="1" Bool
  DrawOutlineConnectionShapes="0" Bool
  DrawSlashConnectionShapes="0" Bool
  DrawBackslashConnectionShapes="0" Bool
/>

<Bar
  ShowShadow="1" Bool
  ShadowColor="#a0a0a0" Color
  OutlineColor="#000000" Color
  ShowOutline="1" Bool
/>

<Area
  Transparency="0" UINT ( 0-255 ) 255 is fully transparent, 0 is opaque
  OutlineColor="#000000" Color
  ShowOutline="1" Bool
/>

<CandleStick
  FillHighClose="0" Bool. If 0, the body is left empty. If 1, FillColorHighClose is used for the candle body
  FillColorHighClose="#ffffff" Color. For the candle body when close > open
  FillHighOpenWithSeriesColor="1" Bool. If true, the series color is used to fill the candlebody when open > close
  FillColorHighOpen="#000000" Color. For the candle body when open > close and FillHighOpenWithSeriesColor is false
/>

<Colors User-defined color scheme: By default this element is empty except for the style and has no Color attributes
  UseSubsequentColors ="1" Boolean. If 0, then color in overlay is used. If 1, then subsequent colors from previous chart layer is used
  Style="User" Possible values are: "Default", "Grayscale", "Colorful", "Pastel", "User"
  Colors="#52aca0" Color: only added for user defined color set
  Colors1="#d3c15d" Color: only added for user defined color set
  Colors2="#8971d8" Color: only added for user defined color set
  ...
  ColorsN="" Up to ten colors are allowed in a set: from Colors to Colors9
</Colors>

<Pie
  ShowLabels="1" Bool
  OutlineColor="#404040" Color
  ShowOutline="1" Bool
  StartAngle="0." Double
  Clockwise="1" Bool
  Draw2dHighlights="1" Bool
  Transparency="0" Int (0 to 255: 0 is opaque, 255 is fully transparent)
  DropShadowColor="#c0c0c0" Color
  DropShadowSize="5.%" PercentOrPixel
  PieHeight="10.%" PercentOrPixel. Pixel values might be different in the result because of 3d tilting
  Tilt="40.0" Double (10 to 90: The 3d tilt in degrees of a 3d pie)
  ShowDropShadow="1" Bool

```

```

ChartToLabelMargin="10.%" PercentOrPixel
AddValueToLabel="0" Bool
AddPercentToLabel="0" Bool
AddPercentToLabels_DecimalDigits="0" UINT ( 0 - 2 )
>
<LabelFont
  Color="#000000"
  Name="Arial"
  Bold="0"
  Italic="0"
  Underline="0"
  MinFontHeight="10. pt"
  Size="4.%" />
</Pie>

<XY>
  <XAxis Axis
    AutoRange="1" Bool
    AutoRangeIncludesZero="1" Bool
    RangeFrom="0. " Double: manual range
    RangeTill="1. " Double : manual range
    LabelToAxisMargin="3.%" PercentOrPixel
    AxisLabel="" String
    AxisColor="#000000" Color
    AxisGridColor="#e6e6e6" Color
    ShowGrid="1" Bool
    UseAutoTick="1" Bool
    ManualTickInterval="1. " Double
    AxisToChartMargin="0. px" PercentOrPixel
    TickSize="3. px" PercentOrPixel
    ShowTicks="1" Bool
    ShowValues="1" Bool
    AxisPosition="LeftOrBottom" Enums: "LeftOrBottom", "RightOrTop",
    "AtValue"
    AxisPositionAtValue = "0" Double
  >
  <ValueFont
    Color="#000000"
    Name="Tahoma"
    Bold="0"
    Italic="0"
    Underline="0"
    MinFontHeight="10. pt"
    Size="3.%" />
  </XAxis>
  <YAxis Axis (same as for XAxis)
    AutoRange="1"
    AutoRangeIncludesZero="1"
    RangeFrom="0. "
    RangeTill="1. "
    LabelToAxisMargin="3.%"
    AxisLabel=""
    AxisColor="#000000"
    AxisGridColor="#e6e6e6"
    ShowGrid="1"
    UseAutoTick="1"
    ManualTickInterval="1. "
    AxisToChartMargin="0. px"
    TickSize="3. px"
    ShowTicks="1" Bool
    ShowValues="1" Bool

```

```

    AxisPosition="LeftOrBottom" Enums: "LeftOrBottom", "RightOrTop",
"AxisSize"
    AxisPositionAtValue = "0" Double
    >
    <ValueFont
        Color="#000000"
        Name="Tahoma"
        Bold="0"
        Italic="0"
        Underline="0"
        MinFontHeight="10. pt"
        Size="3. %"/>
    </YAxis>
</XY>

<XY3d
    AxisAutoSize="1" Bool: If false, XSize and YSize define the aspect ratio of x and y
axis. If true, aspect ratio is equal to chart window
    XSize="100. %" PercentOrPixel. Pixel values might be different in the result because
of 3d tilting and zooming to fit chart
    YSize="100. %" PercentOrPixel. Pixel values might be different in the result because
of 3d tilting and zooming to fit chart
    SeriesMargin="30. %" PercentOrPixel. Pixel values might be different in the result
because of 3d tilting and zooming to fit chart
    Tilt="20. " Double. -90 to +90 degrees
    Rot="20. " Double. -359 to +359 degrees
    FoV="50. "> Double. Field of view: 1-120 degree
    >
    <ZAxis
        AutoRange="1"
        AutoRangeIncludesZero="1"
        RangeFrom="0. "
        RangeTill="1. "
        LabelToAxisMargin="3. %"
        AxisLabel=""
        AxisColor="#000000"
        AxisGridColor="#e6e6e6"
        ShowGrid="1"
        UseAutoTick="1"
        ManualTickInterval="1. "
        AxisToChartMargin="0. px"
        TickSize="3. px" >
        <ValueFont
            Color="#000000"
            Name="Tahoma"
            Bold="0"
            Italic="0"
            Underline="0"
            MinFontHeight="10. pt"
            Size="3. %"/>
        </ZAxis>
</XY3d>

<Gauge
    MinVal="0. " Double
    MaxVal="100. " Double
    MinAngle="225" UINT: -359-359
    SweepAngle="270" UINT: 1-359
    BorderToTick="1. %" PercentOrPixel
    MajorTickWidth="3. px" PercentOrPixel
    MajorTickLength="4. %" PercentOrPixel
    MinorTickWidth="1. px" PercentOrPixel

```

```

MinorTickLength="3.%" PercentOrPixel
BorderColor="#a0a0a0" Color
FillColor="#303535" Color
MajorTickColor="#a0c0b0" Color
MinorTickColor="#a0c0b0" Color
BorderWidth="2.%" PercentOrPixel
NeedleBaseWidth="1.5%" PercentOrPixel
NeedleBaseRadius="5.%" PercentOrPixel
NeedleColor="#f00000" Color
NeedleBaseColor="#141414" Color
TickToTickValueMargin="5.%" PercentOrPixel
MajorTickStep="10." Double
MinorTickStep="5." Double
RoundGaugeBorderToColorRange="0.%" PercentOrPixel
RoundGaugeColorRangeWidth="6.%" PercentOrPixel
BarGaugeRadius="5.%" PercentOrPixel
BarGaugeMaxHeight="20.%" PercentOrPixel
RoundGaugeNeedleLength="45.%" PercentOrPixel
BarGaugeNeedleLength="3.%" PercentOrPixel
>
<TicksFont
  Color="#a0c0b0"
  Name="Tahoma"
  Bold="0"
  Italic="0"
  Underline="0"
  MinFontHeight="10.pt"
  Size="4.%"
/>
<ColorRanges> User-defined color ranges. By default empty with no child element
entries
  <Entry
    From="50." Double
    FillWithColor="1" Bool
    Color="#00ff00" Color
  />
  <Entry
    From="50.0"
    FillWithColor="1"
    Color="#ff0000"
  />
  ...
</ColorRanges>
</Gauge>
</chart-config>

```

### Example: Chart Functions

The example XSLT document below shows how [Altova extension functions for charts](#) can be used. Given further below are an XML document and a screenshot of the output image generated when the XML document is processed with the XSLT document using the Altova XSLT 2.0 Engine.

**Note:** Chart functions are supported only in the **Enterprise and Reporting Editions** of Altova products.

**Note:** For more information about how chart data tables are created, see the documentation of Altova's [XMLSpy](#) and [StyleVision](#) products.



**XSLT document**

This XSLT document (*listing below*) uses Altova chart extension functions to generate a pie chart. It can be used to process the XML document listed further below.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:altovaext="http://www.altova.com/xslt-extensions"
  exclude-result-prefixes="#all">
  <xsl:output version="4.0" method="html" indent="yes" encoding="UTF-8"/>
  <xsl:template match="/">
    <html>
      <head>
        <title>
          <xsl:text>HTML Page with Embedded Chart</xsl:text>
        </title>
      </head>
      <body>
        <xsl:for-each select="/Data/Region[1]">
          <xsl:variable name="extChartConfig" as="item()*">
            <xsl:variable name="ext-chart-settings" as="item()*">
              <chart-config>
                <General>
                  SettingsVersion="1"
                  ChartKind="Pie3d"
                  BKColor="#ffffff"
                  ShowBorder="1"
                  PlotBorderColor="#000000"
                  PlotBKColor="#ffffff"
                  Title="{@id}"
                  ShowLegend="1"
                  OutsideMargin="3.2%"
                  TitleToPlotMargin="3.%"
                  LegendToPlotMargin="6.%"
                >
                <TitleFont>
                  Color="#023d7d"
                  Name="Tahoma"
                  Bold="1"
                  Italic="0"
                  Underline="0"
                  MinFontHeight="10.pt"
                  Size="8.%" />
                </General>
              </chart-config>
            </xsl:variable>
            <xsl:sequence select="
altovaext:create-chart-config-from-xml( $ext-chart-settings )"/>
          </xsl:variable>
          <xsl:variable name="chartDataSeries" as="item()*">
            <xsl:variable name="chartDataRows" as="item()*">
              <xsl:for-each select="(Year)">
                <xsl:sequence select="
altovaext:create-chart-data-row( (@id), (.) )"/>
              </xsl:for-each>
            </xsl:variable>
            <xsl:variable name="chartDataSeriesNames" as="xs:string*"
select=" ( ("Series 1"), &apos;&apos; ) [1]"/>
            <xsl:sequence
select="altovaext:create-chart-data-series-from-rows(
$chartDataSeriesNames, $chartDataRows )"/>
          </xsl:variable>
          <xsl:variable name="ChartObj" select="altovaext:create-chart(
$extChartConfig, ( $chartDataSeries), false() )"/>

```

```

        <xsl:variable name="sChartFileName" select="' mychart1.png' "/>
        
      </xsl:for-each>
    </body>
  </html>
</xsl:template>
</xsl:stylesheet>

```

### XML document

This XML document can be processed with the XSLT document above. Data in the XML document is used to generate the pie chart shown in the screenshot below.

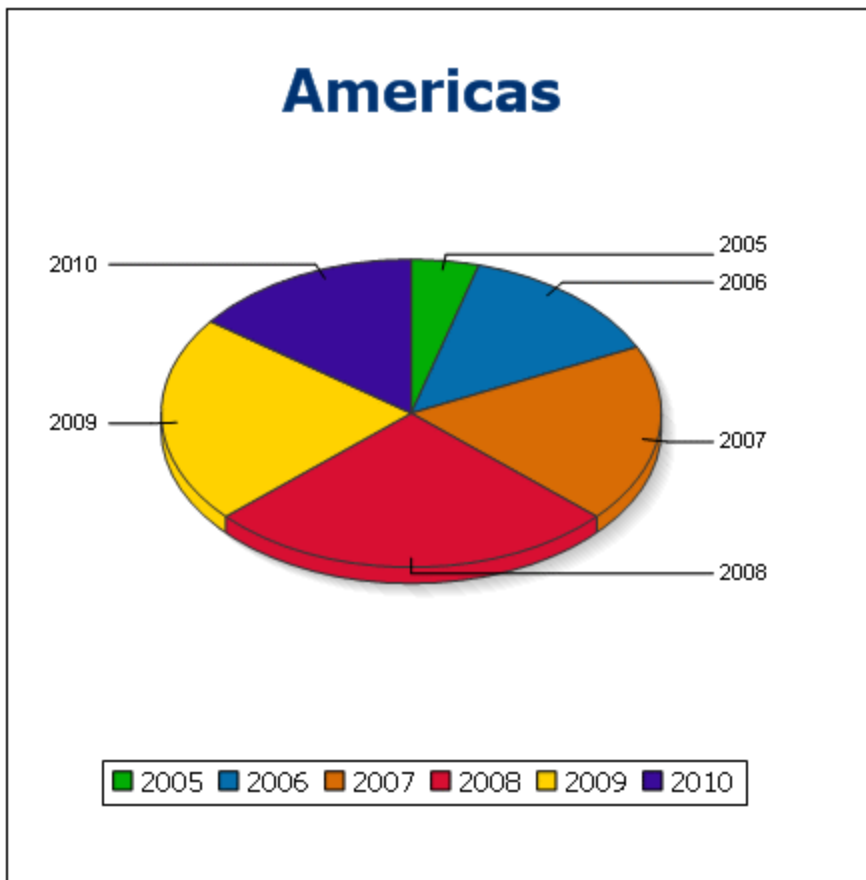
```

<?xml version="1.0" encoding="UTF-8"?>
<Data xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="YearlySales.xsd">
  <ChartType>Pie Chart 2D</ChartType>
  <Region id="Americas">
    <Year id="2005">30000</Year>
    <Year id="2006">90000</Year>
    <Year id="2007">120000</Year>
    <Year id="2008">180000</Year>
    <Year id="2009">140000</Year>
    <Year id="2010">100000</Year>
  </Region>
  <Region id="Europe">
    <Year id="2005">50000</Year>
    <Year id="2006">60000</Year>
    <Year id="2007">80000</Year>
    <Year id="2008">100000</Year>
    <Year id="2009">95000</Year>
    <Year id="2010">80000</Year>
  </Region>
  <Region id="Asia">
    <Year id="2005">10000</Year>
    <Year id="2006">25000</Year>
    <Year id="2007">70000</Year>
    <Year id="2008">110000</Year>
    <Year id="2009">125000</Year>
    <Year id="2010">150000</Year>
  </Region>
</Data>

```

### Output image

The pie chart show below is generated when the XML document listed above is processed with the XSLT document.





## Chapter 4

---

### License Information

## 4 License Information

This section contains:

- Information about the [distribution of this software product](#)
- Information about the [intellectual property rights](#) related to this software product
- The [Altova Developer 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.

## 4.1 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](http://www.altova.com) (i) in HTML format for online browsing, and (ii) in PDF format for download (and to print if you prefer to have the documentation on paper).

### 30-day evaluation period

After downloading this product, you can evaluate it for a period of up to 30 days free of charge. About 20 days into this evaluation period, the software will start to remind you that it has not yet been licensed. The reminder message will be displayed once each time you start the application. If you would like to continue using the program after the 30-day evaluation period, you have to purchase an [Altova Developer License Agreement](#), which is delivered in the form of a key-code that you enter into the Software Activation dialog to unlock the product. You can purchase your license at the online shop at the [Altova website](#).

### Helping Others within Your Organization to Evaluate the Software

If you wish to distribute the evaluation version within your company network, or if you plan to use it on a PC that is not connected to the Internet, you may only distribute the Setup programs, provided that they are not modified in any way. Any person that accesses the software installer that you have provided, must request their own 30-day evaluation license key code and after expiration of their evaluation period, must also purchase a license in order to be able to continue using the product.

For further details, please refer to the [Altova Developer License Agreement](#) at the end of this section.

## 4.2 Software Activation and License Metering

As part of Altova's Software Activation, the software may use your internal network and Internet connection for the purpose of transmitting license-related data at the time of installation, registration, use, or update to an Altova-operated license server and validating the authenticity of the license-related data in order to protect Altova against unlicensed or illegal use of the software and to improve customer service. Activation is based on the exchange of license related data such as operating system, IP address, date/time, software version, and computer name, along with other information between your computer and an Altova license server.

Your Altova product has a built-in license metering module that further helps you avoid any unintentional violation of the End User License Agreement. Your product is licensed either as a single-user or multi-user installation, and the license-metering module makes sure that no more than the licensed number of users use the application concurrently.

This license-metering technology uses your local area network (LAN) to communicate between instances of the application running on different computers.

### Single license

When the application starts up, as part of the license metering process, the software sends a short broadcast datagram to find any other instance of the product running on another computer in the same network segment. If it doesn't get any response, it will open a port for listening to other instances of the application.

### Multi license

If more than one instance of the application is used within the same LAN, these instances will briefly communicate with each other on startup. These instances exchange key-codes in order to help you to better determine that the number of concurrent licenses purchased is not accidentally violated. This is the same kind of license metering technology that is common in the Unix world and with a number of database development tools. It allows Altova customers to purchase reasonably-priced concurrent-use multi-user licenses.

We have also designed the applications so that they send few and small network packets so as to not put a burden on your network. The TCP/IP ports (2799) used by your Altova product are officially registered with the IANA (see <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.



## 4.3 Intellectual Property Rights

The Altova Software and any copies that you are authorized by Altova to make are the intellectual property of and are owned by Altova and its suppliers. The structure, organization and code of the Software are the valuable trade secrets and confidential information of Altova and its suppliers. The Software is protected by copyright, including without limitation by United States Copyright Law, international treaty provisions and applicable laws in the country in which it is being used. Altova retains the ownership of all patents, copyrights, trade secrets, trademarks and other intellectual property rights pertaining to the Software, and that Altova's ownership rights extend to any images, photographs, animations, videos, audio, music, text and "applets" incorporated into the Software and all accompanying printed materials. Notifications of claimed copyright infringement should be sent to Altova's copyright agent as further provided on the Altova Web Site.

Altova software contains certain Third Party Software that is also protected by intellectual property laws, including without limitation applicable copyright laws as described in detail at [http://www.altova.com/legal\\_3rdparty.html](http://www.altova.com/legal_3rdparty.html).

All other names or trademarks are the property of their respective owners.

## 4.4 License Agreement

### THIS IS A LEGAL DOCUMENT -- RETAIN FOR YOUR RECORDS ALTOVA® DEVELOPER LICENSE AGREEMENT

For AltovaXML® Reporting Edition Software  
And AltovaXML® Community Edition Software

Licensor:

Altova GmbH  
Rudolfplatz 13a/9  
A-1010 Wien  
Austria

#### Important – Read Carefully. Notice to User

This Altova Developer License Agreement for AltovaXML® (“DLAXML”) governs your right to use, reproduce, bundle, integrate, and distribute the (i) AltovaXML Reporting Edition Software (“AXRE”) and (ii) the AltovaXML Community Edition Software (“AXCE”) (collectively hereinafter referenced as the “the AltovaXML Software”). Your license rights depend on the specific software edition that you have licensed as the editions have different rights and restrictions applicable to them as set forth in detail below. This DLAXML is a legal document between you and Altova GmbH (“Altova”). It is important that you read this document before using the Altova-provided 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, installing the AltovaXML Software, including the AltovaXML Software, distributing the AltovaXML Software or otherwise using the AltovaXML Software, you agree to be bound by the terms of this DLAXML as well as the Altova Privacy Policy (“Privacy Policy”) including, without limitation, the warranty disclaimers, limitation of liability, data use and termination provisions below. 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 or distribute the AltovaXML Software, and you must destroy any downloaded copies of the AltovaXML Software in your possession or control. Please go to our Web site at <http://www.altova.com/ALTOVADLAXML> to download and print a copy of this DLAXML for your files and <http://www.altova.com/privacy> to review the Privacy Policy.

#### 1. AltovaXML Reporting Edition (“AXRE”) Software Terms and Conditions

(a) **License Grant.** Upon your acceptance of this DLAXML, Altova grants you a non-exclusive, non-transferable (except as provided below), limited license (without the right to grant sublicenses) to; (i) install and use one copy of AXRE Software on compatible personal computer(s) or workstation(s) up to the Permitted Number of computers, including the right to develop your own software applications that include the AXRE Software and Documentation. The Permitted Number of computers and/or users shall be determined and specified at such time as you elect to purchase a license for the software. You may install one copy of the AXRE Software on a computer file server within your internal network solely for the purpose of downloading and installing this AXRE Software onto other computers within your internal network up to the Permitted Number of computers in a commercial environment only. No other network use is permitted, including without limitation using the AXRE 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 AXRE software through a valid license from Altova; and (ii) reproduce the AXRE Software and/or Documentation and distribute the AXRE Software in executable form, subject to the Distribution

Restrictions and Requirements below, to third parties electronically or via download from your website or on physical media in conjunction with a software application developed by you incorporating the AXRE Software. Notwithstanding anything to the contrary, and for avoidance of doubt, you may provide access on your network to a web service or other service that you have developed that utilizes the AXRE Software features or functions in a custom application that you have developed and displays a report or other results data generated by your service in response to and based on data submitted by a user accessing the service.

You may make one backup and one archival copy of the AXRE 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 AXRE software as provided in this DLAXML.

**(b) Distribution Restrictions and Requirements.** In addition to the restrictions and obligations provided in other sections of this DLAXML, your license to distribute the AXRE Software and/or Documentation is further subject to all of the following restrictions: **(i)** the AXRE Software and/or Documentation component of your software application shall only be licensed and not sold and all end users are required to purchase a license from Altova or an authorized reseller, in order to use the AXRE Software component of your software application; **(ii)** you may not make the AXRE Software and/or Documentation available as a stand-alone product but only as a component of or in conjunction with your software application; **(iii)** you must use the Setup Program for AXRE provided by Altova AS IS and may not impair, alter or remove Altova's DLAXML (which will appear in the installation process and which an end user must accept in order to be able to install or operate the AXRE software) or any other files; and **(iv)** other Altova products cannot be distributed or used under this DLAXML.

**(c) Key Codes, Upgrades and Updates.** You will receive a purchase key code when you elect to purchase AXRE software license(s) from either Altova GmbH or an authorized reseller. The purchase key code will enable you to activate the software. You may not re-license, reproduce or distribute any key code except with the express written permission of Altova. If the AXRE software that you have licensed is an upgrade or an update, then the latest update or upgrade that you download and install replaces all or part of the Software previously licensed. The update or upgrade and the associated license keys, as applicable, does not constitute the granting of a second license to the software in that you may not use the upgrade or updated copy in addition to the copy of the software that it is replacing and whose license has terminated.

**(d) Support and Maintenance.** Altova offers "Support & Maintenance Package(s)" ("SMP") for the AXRE Software that you have licensed. 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:

(i) 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 AXRE 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 1(d)), 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.

(ii) 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 AXRE 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 a 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.

(iii) 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 AXRE 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 AXRE Software fails to perform. You shall use reasonable efforts to cooperate in diagnosis or study of errors. Altova may include error corrections in maintenance releases, updates, or new major releases of the software. Altova is not obligated to fix errors that are immaterial. Immaterial errors are those that do not significantly impact use of the software, as determined by Altova in its sole discretion. Whether or not you have purchased the Support & Maintenance Package, technical support only covers issues or questions resulting directly out of the operation of the AXRE software and Altova will not provide you with generic consultation, assistance, or advice under any circumstances.

(iv) Updating the AXRE Software may require the updating of software not covered by this DLAXML before installation. Updates of the operating system and application software not specifically covered by this DLAXML are your responsibility and will not be provided by Altova under this DLAXML. Altova's obligations under this Section are contingent upon your proper use of the AXRE Software and your compliance with the terms and conditions of this DLAXML at all times. Altova shall be under no obligation to provide the above technical support if, in Altova's opinion, the AXRE Software has failed due to the following conditions: (a) damage caused by the relocation of the software to another location or CPU; (b) alterations, modifications or attempts to change the software without Altova's written approval; (c) causes external to the software, such as natural disasters, the failure or fluctuation of electrical power, or computer equipment failure; (d) your failure to maintain the software at Altova's specified release level; or (e) use of the software with other software without Altova's prior written approval. It will be your sole responsibility to: (a) comply with all Altova-specified operating and troubleshooting procedures and then notify Altova immediately of AXRE Software malfunction and provide Altova with complete information thereof; (b) provide for the security of your confidential information; and (c) establish and maintain backup systems and procedures necessary to reconstruct lost or altered files, data or programs.

(e) **Limited Warranty.** Altova warrants to the person or entity that first purchases a license for use of the AXRE software pursuant to the terms of this DLAXML 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 1(d) 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 AXRE 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 AXRE 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 AXRE 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 Software. THE FOREGOING LIMITED WARRANTY AND REMEDIES STATE THE SOLE AND EXCLUSIVE REMEDIES FOR ALTOVA'S OR ITS SUPPLIERS' 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.

**(f) Limitation of Liability and Infringement Claims.** 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 AXRE 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 DLAXML SHALL BE LIMITED TO THE AMOUNT ACTUALLY PAID BY YOU FOR THE AXRE 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 DLAXML between Altova and you. 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 AXRE 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 this Section 1(f) of this Agreement except as otherwise expressly provided. You must notify Altova in writing of any Claim within ten (10) business days after you first receive notice of the Claim, and you shall provide to Altova at no cost such assistance and cooperation as Altova may reasonably request from time to time in connection with the defense of the Claim. Altova shall have sole control over any Claim (including, without limitation, the selection of counsel and the right to settle on your behalf on any terms Altova deems desirable in the sole exercise of its discretion). You may, at your sole cost, retain separate counsel and participate in the defense or settlement negotiations. Altova shall pay actual damages, costs, and attorney fees awarded against you (or payable by you pursuant to a settlement agreement) in connection with a Claim to the extent such direct

damages and costs are not reimbursed to you by insurance or a third party, to an aggregate maximum equal to the purchase price of the AXRE Software. If the AXRE 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 AXRE 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 DLAXML 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.

**(g) Applicable DLAXML Terms.** The terms and conditions set forth in Sections 1 and 3 apply to the AXRE Software.

## **2. AltovaXML Community Edition ("AXCE") Software Terms and Conditions**

**(a) License Grant.** Upon your acceptance of this DLAXML, Altova grants you a non-exclusive, non-transferable (except as provided below), limited license (without the right to grant sublicenses) to **develop software applications that include the AXCE Software and/or Documentation, reproduce the AXCE Software and/or Documentation, and distribute the AXCE Software in executable form, subject to the Distribution Restrictions and Requirements below, to third parties electronically or via download from your website or on physical media in conjunction with a software application developed by you incorporating the AXCE Software. You may install the AXCE Software on a server within your network for the purpose of downloading and installing the Software (to an unlimited number of client computers on your internal network).**

**(b) Distribution Restrictions and Requirements.** In addition to the restrictions and obligations provided in other sections of this DLAXML, your license to distribute the AXCE Software and/or Documentation is further subject to all of the following restrictions: **(i) the AXCE Software and/or Documentation shall only be licensed and not sold; (ii) you may not make the AXCE Software and/or Documentation available as a standalone product and if distributed as part of a product bundle you may charge for the product bundle provided that you license such product bundle at the same or lower fee at which you license any reasonably equivalent product bundle which does not include the AXCE Software; (iii) you must use the Setup Program for AXCE software provided by Altova AS IS and may not impair, alter or remove Altova's DLAXML (which will appear in the installation process and which an end user must accept in order to be able to install or operate the AXCE software) or any other files; and (iv) other Altova products cannot be distributed or used under this DLAXML.**

**(c) Warranty Disclaimer.** THE AXCE SOFTWARE IS PROVIDED TO YOU FREE OF CHARGE, AND ON AN "AS-IS" BASIS. ALTOVA PROVIDES NO WARRANTIES FOR THE AXCE SOFTWARE. TO THE MAXIMUM EXTENT PERMITTED BY LAW, ALTOVA AND ITS SUPPLIERS DISCLAIM ALL WARRANTIES AND REPRESENTATIONS, WHETHER EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, SATISFACTORY QUALITY, INFORMATIONAL CONTENT, OR ACCURACY, QUIET ENJOYMENT, TITLE, AND NON-INFRINGEMENT. ALTOVA DOES NOT WARRANT THAT THE AXCE SOFTWARE IS ERROR-FREE OR WILL OPERATE WITHOUT INTERRUPTION. IF MANDATORILY APPLICABLE LAW REQUIRES ANY WARRANTIES WITH RESPECT TO THE AXCE SOFTWARE, ALL SUCH WARRANTIES ARE LIMITED IN DURATION TO THIRTY (30) -DAYS FROM THE DATE OF INSTALLATION OR BEGIN OF USE, WHATEVER IS THE EARLIER. SOME STATES OR JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF

IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU. THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS AND YOU MAY ALSO HAVE OTHER LEGAL RIGHTS THAT VARY FROM STATE TO STATE OR FROM JURISDICTION TO JURISDICTION. YOU AGREE THAT YOU ARE SOLELY RESPONSIBLE FOR THE ACCURACY AND ADEQUACY OF THE ACDE OR ACBE 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 ACDE OR ACBE SOFTWARE IN YOUR USE.

**(d) Limitation of Liability.** TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, 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 AUTHENTIC SOFTWARE, THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES, OR ANY PROVISION OF THIS DLAXML, EVEN IF ALTOVA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. WHERE LEGALLY, LIABILITY CANNOT BE EXCLUDED, BUT MAY BE LIMITED, ALTOVA'S LIABILITY AND THAT OF ITS SUPPLIERS SHALL BE LIMITED TO THE SUM OF FIVE DOLLARS (USD \$5.00) IN TOTAL. 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 AND THAT OF ITS SUPPLIERS SHALL BE LIMITED TO THE GREATEST EXTENT PERMITTED BY LAW. THE FOREGOING LIMITATIONS ON LIABILITY ARE INTENDED TO APPLY TO THE WARRANTIES AND DISCLAIMERS ABOVE AND ALL OTHER ASPECTS OF THIS DLAXML

**(e) Support.** Altova is not obliged to provide technical support with respect to the AXCE software.

**(f) Applicable DLAXML Terms.** The terms and conditions set forth in Sections 2 and 3 apply to the AXCE software.

### **3. AltovaXML Software (AXRE and AXCE) Software Terms and Conditions**

The terms set forth in Section 3 are applicable to the AXRE and AXCE software licenses and are in addition to the specific terms applicable to those software licenses.

**(a) Title.** This DLAXML gives you a limited license to reproduce and distribute the **AltovaXML Software and/or Documentation as set forth above. Altova and its suppliers retain all right, title and interest, including all copyright and intellectual property rights, in and to, the AltovaXML Software and/or Documentation and all copies thereof.** Title to the AltovaXML Software is not transferred to you. Ownership of all copies of the AltovaXML Software and of copies made by you is vested in Altova, subject to the rights of use or distribution, as applicable, granted to you in this DLAXML. All rights not specifically granted in this DLAXML are reserved by Altova.

**(b) Acknowledgement of Altova's Rights.** You acknowledge that the AltovaXML 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 AltovaXML Software are the valuable trade secrets and confidential information of Altova and its suppliers. The AltovaXML 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 AltovaXML Software, and that Altova's ownership rights extend to any images, photographs, animations, videos, audio, music, text and "applets" incorporated into the AltovaXML Software

and all accompanying printed materials. You will take no actions which adversely affect Altova's intellectual property rights in the AltovaXML 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 AltovaXML Software, and such use of any trademark does not give you any right of ownership in that trademark. XMLSpy®, Authentic®, StyleVision®, MapForce®, UModel®, DatabaseSpy®, DiffDog®, SchemaAgent®, SemanticWorks®, MissionKit®, Markup Your Mind®, , and Altova® are trademarks of Altova GmbH (registered in numerous countries). Unicode and the Unicode Logo are trademarks of Unicode, Inc. Windows, Windows XP, Windows Vista, and Windows 7 are trademarks of Microsoft. W3C, CSS, DOM, MathML, RDF, XHTML, XML and XSL are trademarks (registered in numerous countries) of the World Wide Web Consortium (W3C); marks of the W3C are registered and held by its host institutions, MIT, INRIA and Keio. Except as expressly stated above, this DLAXML does not grant you any intellectual property rights in the AltovaXML Software. Notifications of claimed copyright infringement should be sent to Altova's copyright agent as further provided on the Altova Web Site.

**(c) Common Restrictions.**

(i) 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 AltovaXML 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 AltovaXML 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 AltovaXML Software. Requests for information from users in the European Union with respect to the above should be directed to the Altova Customer Support Department. You may not loan, rent, lease, sublicense, distribute or otherwise transfer all or any portion of the AltovaXML Software to third parties except to the limited extent expressly provided in this DLAXML.

(ii) You may not copy, distribute, or make derivative works of the AltovaXML Software except as expressly set forth above, and any copies that you are permitted to make pursuant to this DLAXML must contain the same copyright, patent and other intellectual property markings that appear on or in the AltovaXML Software. You may not modify, adapt or translate the AltovaXML Software. You may not, directly or indirectly, encumber or suffer to exist any lien or security interest on the AltovaXML Software; knowingly take any action that would cause the AltovaXML Software to be placed in the public domain; or use the AltovaXML Software in any computer environment not specified in this DLAXML. You will comply with applicable law and Altova's instructions regarding the use of the AltovaXML Software. You agree to notify your employees and agents who may have access to the AltovaXML Software of the restrictions contained in this DLAXML and to ensure their compliance with these restrictions. You may not alter or modify the AltovaXML Software or create a new installer for the AltovaXML Software. You agree to indemnify, hold harmless, and defend Altova from and against any claims or lawsuits, including attorney's fees that arise or result from your use or distribution of the Software and/or Documentation

**(d) AltovaXML Software Activation, Updates, Metering and Data Use.**

(i) Altova has a built-in license metering module that helps you to avoid any unintentional violation of this DLAXML. Altova may use your internal network for license metering between installed versions of the AltovaXML Software. **Altova's AltovaXML Software may use your internal network and Internet connection for the purpose of transmitting**



license-related data at the time of installation, registration, use, or update to an Altova-operated license server and validating the authenticity of the license-related data in order to protect Altova against unlicensed or illegal use of the AltovaXML Software and to improve customer service. Activation is based on the exchange of license related data between your computer and the Altova license server. You agree that Altova may use these measures and you agree to follow any applicable requirements. You further agree that use of license key codes that are not or were not generated by Altova and lawfully obtained from Altova or an authorized reseller as part of an effort to activate or use the AltovaXML Software violates Altova's intellectual property rights as well as the terms of this DLAXML. You agree that efforts to circumvent or disable Altova's copyright protection mechanisms or license management mechanism violate Altova's intellectual property rights as well as the terms of this DLAXML. Altova expressly reserves the rights to seek all available legal and equitable remedies to prevent such actions and to recover lost profits, damages and costs.

(ii) 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.

(iii) 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 DLAXML. By your acceptance of the terms of this DLAXML or use of the AltovaXML Software, you authorize the collection, use and disclosure of information collected by Altova for the purposes provided for in this DLAXML and/or the Privacy Policy as revised from time to time.

(iv) Notice to European Users. Please note that the information as described in paragraph 3(d) above may be transferred outside of the European Economic Area, for purposes of processing, analysis, and review, by Altova, Inc. a company located in Beverly, Massachusetts, U.S.A., or its subsidiaries or Altova's subsidiaries or divisions, or authorized partners, located worldwide. You are advised that the United States uses a sectoral model of privacy protection that relies on a mix of legislation, governmental regulation, and self-regulation. You are further advised that the Council of the European Union has found that this model does not provide "adequate" privacy protections as contemplated by Article 25 of the European Union's Data Directive. (Directive 95/46/EC, 1995 O.J. (L 281) 31). Article 26 of the European Union's Data Directive allows for transfer of personal data from the European Union to a third country if the individual has unambiguously given his consent to the transfer of personal information, regardless of the third country's level of protection. By agreeing to this Software License Agreement, you consent to the transfer of all such information to the United States and the processing of that information as described in this Software License Agreement and the Privacy Policy.

**(e) Disclaimer.** THE ALTOVAXML SOFTWARE IS NEITHER GUARANTEED NOR WARRANTED TO BE ERROR-FREE NOR SHALL ANY LIABILITY BE ASSUMED BY ALTOVA IN THIS RESPECT. NOTWITHSTANDING ANY SUPPORT FOR ANY TECHNICAL STANDARD, THE ALTOVAXML SOFTWARE IS NOT INTENDED FOR USE IN OR IN CONNECTION WITH, WITHOUT LIMITATION, THE OPERATION OF NUCLEAR FACILITIES, AIRCRAFT NAVIGATION, COMMUNICATION SYSTEMS, AIR TRAFFIC CONTROL EQUIPMENT, MEDICAL DEVICES OR LIFE SUPPORT SYSTEMS, MEDICAL OR HEALTH CARE APPLICATIONS, OR OTHER APPLICATIONS WHERE THE FAILURE OF THE ALTOVAXML SOFTWARE OR ERRORS IN DATA PROCESSING COULD LEAD TO DEATH, PERSONAL INJURY OR SEVERE PHYSICAL OR ENVIRONMENTAL DAMAGE. YOU AGREE THAT YOU ARE SOLELY RESPONSIBLE FOR THE ACCURACY AND ADEQUACY OF THE ALTOVAXML SOFTWARE AND ANY DATA GENERATED OR PROCESSED BY THE SOFTWARE FOR YOUR INTENDED USE AND YOU WILL DEFEND, INDEMNIFY AND HOLD ALTOVA, ITS OFFICERS AND EMPLOYEES HARMLESS FROM ANY 3RD PARTY CLAIMS, DEMANDS, OR SUITS THAT ARE BASED UPON THE ACCURACY AND ADEQUACY OF

THE ALTOVAXML SOFTWARE IN YOUR USE OR ANY DATA GENERATED BY THE ALTOVA XML SOFTWARE IN YOUR USE.

**(f) Restricted Rights Notice and Export Restrictions.** The AltovaXML 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 DLAXML. Manufacturer is Altova GmbH, Rudolfsplatz, 13a/9, A-1010 Vienna, Austria/EU. You may not use or otherwise export or re-export the AltovaXML Software or documentation except as authorized by United States law and the laws of the jurisdiction in which the AltovaXML Software was obtained. In particular, but without limitation, the AltovaXML 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 AltovaXML 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.

**(g) Termination.** Without prejudice to any other rights or remedies of Altova, this DLAXML may be terminated (i) by you giving Altova written notice of termination or (ii) by Altova, at its option, giving you written notice of termination or (iii) Altova giving you written notice of termination if you fail to comply with the terms and conditions of the DLAXML. Upon any termination or expiration of this DLAXML, you must cease all use or distribution of the AltovaXML Software, licensed hereunder, 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 AltovaXML Software remain in your possession or control or available for distribution in conjunction with your software product. The terms and conditions set forth in Sections 1(e)-(f), 2(c)-(d), 3(c)-(i) survive termination of this DLAXML as applicable.

**(h) Third Party Software.** The AltovaXML Software may contain third party software which requires notices and/or additional terms and conditions. Such required third party software notices and/or additional terms and conditions are located at our Website at [http://www.altova.com/legal\\_3rdparty.html](http://www.altova.com/legal_3rdparty.html) and are made a part of and incorporated by reference into this DLAXML. By accepting this DLAXML, you are also accepting the additional terms and conditions, if any, set forth therein.

**(i) General Legal Provisions.** This DLAXML 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 DLAXML 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 DLAXML 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 DLAXML. This DLAXML 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 DLAXML 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 DLAXML. If, for any reason, any provision of this DLAXML is held invalid or otherwise unenforceable, such invalidity or unenforceability shall not affect the remainder of this DLAXML, and this DLAXML 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.

(i) If you are located in the European Union and are using the AltovaXML Software in the European Union and not in the United States, then this DLAXML 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 AltovaXML 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.

(ii) If you are located in the United States or are using the AltovaXML Software in the United States then this DLAXML 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 AltovaXML 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 the Commonwealth of Massachusetts in connection with any such dispute or claim.

(iii) If you are located outside of the European Union or the United States and are not using the AltovaXML Software in the United States, then this DLAXML 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 AltovaXML 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 DLAXML 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.

Last Updated: 2011-01-04



# Index

■

## **.NET extension functions,**

- constructors, 134
- datatype conversions, .NET to XPath/XQuery, 137
- datatype conversions, XPath/XQuery to .NET, 136
- for XSLT and XQuery, 132
- instance methods, instance fields, 135
- overview, 132
- static methods, static fields, 134

## **.NET interface,**

- example code, 82
- features, 4
- object `Altova.AltovaXML.XMLValidator`, 84
- object `Altova.AltovaXML.XQuery`, 96
- object `Altova.AltovaXML.XSLT1`, 86
- object `Altova.AltovaXML.XSLT2`, 91
- object model, 82
- usage, 80, 82

## A

### **Altova Developer License Agreement, 160, 164**

### **Altova extension functions,**

- chart functions (see chart functions), 141
- general functions, 141

### **Altova extensions,**

- chart functions (see chart functions), 141

### **Altova XSLT 1.0 Engine,**

- limitations and implementation-specific behavior, 108

### **Altova XSLT 2.0 Engine,**

- general information about, 111
- information about, 110

### **Altova.AltovaXML.Application object, 82**

### **Altova.AltovaXML.dll, 6, 80**

### **AltovaXML,**

- available functionality, 5
- COM interface features, 4
- command line features, 4
- documentation, 3

- installation, 6
- introduction, 3
- main features of, 4
- package, 4
- system requirements for, 6
- usage of, 10
- user manual, 3

### **AltovaXML.jar, 40**

- and CLASSPATH, 6

### **AltovaXMLLib.dll, 6, 40**

### **atomization of nodes,**

- in XPath 2.0 and XQuery 1.0 evaluation, 118

## B

### **backwards compatibility,**

- of XSLT 2.0 Engine, 111

## C

### **C# example code,**

- for .NET interface, 82

### **C++ example code,**

- for COM interface, 38

### **character entities,**

- in HTML output of XSLT transformation, 108

### **character normalization,**

- in XQuery document, 114

### **Chart functions,**

- chart data structure for, 150
- example, 154
- listing, 146

### **CLASSPATH,**

- and `AltovaXML.jar`, 6

### **collations,**

- in XPath 2.0, 118
- in XQuery document, 114

### **COM interface,**

- Application object, 25
- C++ example code, 38
- example code, 36
- features, 4
- JScript example code, 37
- object model, 24

**COM interface,**

- usage, 22
- Validator interface, 26
- Visual Basic example code, 36
- XQuery interface, 33
- XSLT1 interface, 28
- XSLT2 interface, 30

**COM Server,**

- Releasing, 102

**COM server object,**

- registering AltovaXML as, 6, 11, 23

**com.altova.engines, 40****Command line,**

- features, 4
- for XQuery 1.0 executions, 20
- for XSLT 1.0 transformations, 16
- for XSLT 2.0 transformations, 18
- help, 11
- usage summary, 11
- validation and well-formedness check, 13
- version information from, 11

**Copyright, 163****Copyright information, 160****count() function,**

- in XPath 1.0, 108

**count() function in XPath 2.0,**

- see fn:count(), 118

**D****datatypes,**

- in XPath 2.0 and XQuery 1.0, 118

**deep-equal() function in XPath 2.0,**

- see fn:deep-equal(), 118

**default functions namespace,**

- for XPath 2.0 and XQuery 1.0 expressions, 118
- in XSLT 2.0 stylesheets, 111

**Dispatch Interface,**

- description of, 22

**Documentation,**

- overview of, 7

**Dot NET,**

- see .NET, 80

**E****encoding,**

- in XQuery document, 114

**Engine information, 106****Evaluation period, 160, 161****Examples, 22, 36, 40, 82****Extension functions for XSLT and XQuery, 123****Extension Functions in .NET for XSLT and XQuery,**

- see under .NET extension functions, 132

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

- see under Java extension functions, 124

**Extension Functions in MSXSL scripts, 138****external functions,**

- in XQuery document, 114

**F****fn:base-uri in XPath 2.0,**

- support in Altova Engines, 120

**fn:collection in XPath 2.0,**

- support in Altova Engines, 120

**fn:count() in XPath 2.0,**

- and whitespace, 118

**fn:current-date in XPath 2.0,**

- support in Altova Engines, 120

**fn:current-dateTime in XPath 2.0,**

- support in Altova Engines, 120

**fn:current-time in XPath 2.0,**

- support in Altova Engines, 120

**fn:data in XPath 2.0,**

- support in Altova Engines, 120

**fn:deep-equal() in XPath 2.0,**

- and whitespace, 118

**fn:id in XPath 2.0,**

- support in Altova Engines, 120

**fn:idref in XPath 2.0,**

- support in Altova Engines, 120

**fn:index-of in XPath 2.0,**

- support in Altova Engines, 120

**fn:in-scope-prefixes in XPath 2.0,**

- support in Altova Engines, 120

**fn:last() in XPath 2.0,**

**fn:last() in XPath 2.0**,  
and whitespace, 118

**fn:lower-case in XPath 2.0**,  
support in Altova Engines, 120

**fn:normalize-unicode in XPath 2.0**,  
support in Altova Engines, 120

**fn:position() in XPath 2.0**,  
and whitespace, 118

**fn:resolve-uri in XPath 2.0**,  
support in Altova Engines, 120

**fn:static-base-uri in XPath 2.0**,  
support in Altova Engines, 120

**fn:upper-case in XPath 2.0**,  
support in Altova Engines, 120

**Functionality**,  
of AltovaXML, 5

**functions**,  
see under XSLT 2.0 functions, 113  
XPath 2.0 and XQuery 1.0, 117

## H

**Help**,  
from command line, 11

## I

**implementation-specific behavior**,  
of XSLT 2.0 functions, 113

**implicit timezone**,  
and XPath 2.0 functions, 118

**Installation**,  
of AltovaXML, 6

**Intellectual property rights, 163**

## J

**Java class AltovaXMLFactory**,  
description of, 65

**Java class XMLValidator**,  
description of, 67

**Java class XQuery**,  
description of, 69

**Java class XSLT1**,  
description of, 73

**Java class XSLT2**,  
description of, 76

**Java extension functions**,  
constructors, 128  
datatype conversions, Java to Xpath/XQuery, 131  
datatype conversions, XPath/XQuery to Java, 130  
for XSLT and XQuery, 124  
instance methods, instance fields, 129  
overview, 124  
static methods, static fields, 128  
user-defined class files, 125  
user-defined JAR files, 127

**Java interface**,  
additional documentation, 40  
example code, 40  
features, 4  
setup, 40  
summary of classes, 58, 65  
usage, 40

**Java interface IAltovaXMLEngine**,  
description of, 58

**Java interface IAltovaXMLFactory**,  
description of, 59

**Java interface IExecutable**,  
description of, 60

**Java interface IReleasable**,  
description of, 60

**Java interface IXMLValidator**,  
description of, 61

**Java interface IXQuery**,  
description of, 62

**Java interface IXSLT**,  
description of, 64

**JScript example code**,  
for COM interface, 37

## L

**last() function**,  
in XPath 1.0, 108

**last() function in XPath 2.0**,  
see fn:last(), 118

**Legal information, 160**

**library modules**,

**library modules,**  
in XQuery document, 114

**License, 164**  
information about, 160

**License metering, 162**

## M

**msxsl:script, 138**

## N

**namespaces,**  
in XQuery document, 114  
in XSLT 2.0 stylesheet, 111

## O

**OOXML files, 103**

## P

**position() function,**  
in XPath 1.0, 108

**position() function in XPath 2.0,**  
see fn:position(), 118

## Q

**QName serialization,**  
when returned by XPath 2.0 functions, 120

## R

**Raw Interface,**  
description of, 22

**Registering AltovaXML,**  
as COM server object, 6, 11, 23

**Releasing,**  
COM server, 102

## S

**schema validation of XML document,**  
for XQuery, 114

**schema-awareness,**  
of XPath 2.0 and XQuery Engines, 118

**Scripts in XSLT/XQuery,**  
see under Extension functions, 123

**Standards conformance,**  
of Altova engines, 106  
of Altova XML Validator, 107

## U

**Unregistering AltovaXML as a COM server object, 23**

**Usage,**  
of AltovaXML, 10

## V

**Validation,**  
available functionality, 5  
from command line, 13  
using .NET interface, 84

**Version information,**  
from command line, 11

**Visual Basic example code,**  
for COM interface, 36

## W

**Well-formedness check,**  
from command line, 13  
using .NET interface, 84

**whitespace handling,**  
and XPath 2.0 functions, 118

**whitespace in XML document,**  
handling by Altova XSLT 2.0 Engine, 111



**whitespace nodes in XML document,**  
and handling by XSLT 1.0 Engine, 108

## X

### **XML validation,**

see validation, 84

### **XPath 2.0 functions,**

general information about, 118  
implementation information, 117  
see under fn: for specific functions, 118

### **XPath functions support,**

see under fn: for individual functions, 120

### **XQuery,**

Extension functions, 123

### **XQuery 1.0 Engine,**

information about, 114

### **XQuery 1.0 functions,**

general information about, 118  
implementation information, 117  
see under fn: for specific functions, 118

### **XQuery 1.0 transformations,**

using .NET interface, 96

### **XQuery executions,**

available functionality, 5  
from command line, 20

### **xs:QName,**

also see QName, 120

### **xsl:preserve-space, 108**

### **xsl:strip-space, 108**

### **XSLT,**

Extension functions, 123

### **XSLT 1.0 Engine,**

limitations and implementation-specific behavior, 108

### **XSLT 1.0 transformations,**

from command line, 16  
using .NET interface, 86

### **XSLT 2.0 Engine,**

general information about, 111  
information about, 110

### **XSLT 2.0 functions,**

implementation-specific behavior of, 113  
see under fn: for specific functions, 113

### **XSLT 2.0 stylesheet,**

namespace declarations in, 111

### **XSLT 2.0 transformations,**

from command line, 18  
using .NET interface, 91

### **XSLT transformations,**

available functionality, 5

## Z

### **ZIP files, 103**