Altova RaptorXML 2013

User and Reference Manual

Altova RaptorXML 2013 User & Reference Manual

Published: 2013

Table of Contents

Chapter 1

About RaptorXML

1 About RaptorXML

Altova RaptorXML Development Edition (hereafter also called RaptorXML for short) is Altova's third-generation, hyper-fast XML and XBRL* processor. It has been built to be optimized for the latest standards and parallel computing environments. Designed to be highly cross-platform capable, the engine takes advantage of today's ubiquitous multi-core computers to deliver lightning fast processing of XML and XBRL data.

*** Note:** XBRL processing is available only in RaptorXML+XBRL Server, not in RaptorXML Server or RaptorXML Development Edition.

Editions and operating systems
There are three editions of RaptorXML, each suitable for a different set of requirements. These editions are described in the section Editions and Interfaces. While the Development Edition is available only on the Windows operating system, the two server editions are available for Windows, Linux, and Mac OS X. For more details of system support, see the section System Requirements.

Features and supported specifications
RaptorXML provides XML validation, XSLT transformations, and XQuery executions, each with a wide range of powerful options. See the section Features for a broad list of available functionality and key features. The section Supported Specifications provides a detailed list of the specifications to which RaptorXML conforms. For more information, visit the RaptorXML page at the Altova website.

This documentation
This documentation is delivered with the application and is also available online at the Altova website. Note that the Chrome browser has a limitation that prevents entries in the Table of Contents (TOC) pane expanding when the documentation is opened locally. The TOC in Chrome functions correctly, however, when the documentation is opened from a webserver.

This documentation is organized into the following sections:

- About RaptorXML (this section)
- Setting Up RaptorXML
- Command Line Interface
- Java Interface
- COM/.NET Interface
- XSLT and XQuery Engine Information
- XSLT and XQuery Extension Functions

Last updated: 07/23/2013

1.1 Editions and Interfaces

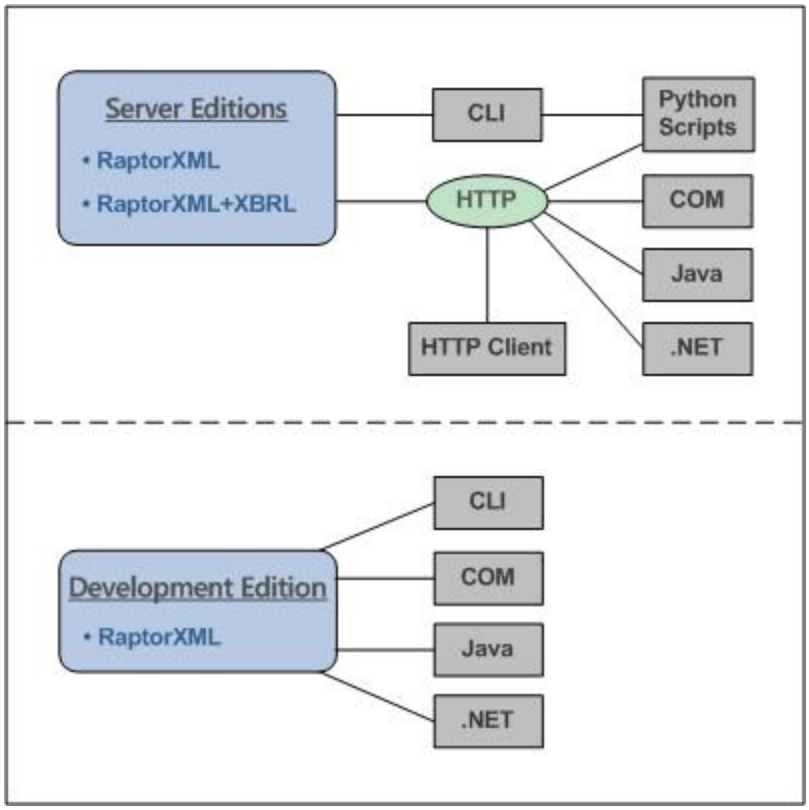
RaptorXML is available in three editions:

- *RaptorXML Server* is a very fast XML processing engine with support for XML, XML Schema, XSLT, XPath, XQuery, and more.
- *RaptorXML+XBRL Server* supports all the features of RaptorXML Server with the additional capability of processing and validating the XBRL family of standards.
- *RaptorXML Development Edition* is available for Windows systems. It can be downloaded as a separate package from the Altova website and can be activated with the license details of any Altova MissionKit products (XMLSpy, MapForce, and StyleVision). Like RaptorXML(+XBRL) Server, it can be used for XML, XSLT, and XQuery validations and transformations, but it has limited functionality in comparison with RaptorXML(+XBRL) Server. The limitations are as follows:
  - Only one instance of of its binary is allowed to run at a given time on a given machine.
  - Supported only on Windows workstations (not on Windows Server, Linux, or Mac OS X operating systems).
  - No support for multiple threads or bulk-processing (so, no multiple file processing).
  - No support for rendering charts.
  - No XBRL support.
  - 32-bit version only.

Interfaces
RaptorXML is accessed via the following interfaces:

- A command line interface (CLI) *(all three editions)*
- A COM interface on Windows systems *(all three editions)*
- A .NET interface on Windows systems *(all three editions)*
- A Java interface on Windows, Linux, and MacOS systems *(all three editions)*
- An HTTP interface that can be accessed by an HTTP client *(server editions only)*
- A Python interface with which Python scripts can access and process document parts via the Python APIs of RaptorXML. Scripts can be submitted via CLI or HTTP *(server editions only)*

The diagram below shows how the two server editions (RaptorXML Server and RaptorXML+XBRL Server) and RaptorXML Development Edition are accessed via their interfaces.



Notice that the COM, Java, and .NET interfaces use the HTTP protocol to connect to the server editions. Python scripts can be submitted to the server editions via the command line and HTTP interfaces.

Command line interface (CLI)
Provides command line usage for XML (and other document) validation, XSLT transformation, and XQuery execution. See the section Command Line for usage information.

HTTP interface
All the functionality of the server editions can be accessed via an HTTP interface. Client requests are made in JSON format. Each request is assigned a job directory on the server, in which output files are saved, Server responses to the client include all relevant information about the job. See the section HTTP Interface.

Python interface
Together with a CLI command or HTTP request, a Python script can be submitted that accesses document/s specified in the command or request. Access to the document is provided by Python APIs for XML, XSD, and XBRL. See the section Python Interface for a description of usage and the APIs.

COM interface
RaptorXML can be used via COM interface, and therefore can be used by applications and scripting languages that support COM. COM interface support is implemented for Raw and Dispatch interfaces. Input data can be provided as files or as text strings in scripts and in application data.

Java interface
RaptorXML functionality is available as Java classes that can be used in Java programs. For example, there are Java classes that provide XML validation, XSLT transformation, and XQuery execution features.

.NET interface
A DLL file is built as a wrapper around RaptorXML and allows .NET users to connect to RaptorXML functionality. RaptorXML provides primary interop assembly signed by Altova. Input data can be provided as files or as text strings in scripts and in application data.

1.2 System Requirements

RaptorXML is supported on the following operating systems:

- Windows Server 2008 R2, or newer
- Windows XP with Service Pack 3, Windows 7, Windows 8, or newer
- Linux (CentOS 6, RedHat 6, Debian 6, and Ubuntu 12.04, or newer)
- Mac OS X 10.7 or newer

RaptorXML Server editions are available for both 32-bit and 64-bit machines. Specifically these are x86 and amd64 (x86-64) instruction-set based cores: Intel Core i5, i7, XEON E5. RaptorXML Development Edition is available for 32-bit machines only.

To use RaptorXML 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.

1.3 Features

RaptorXML provides the functionality listed below. Most functionality is common to command line usage and COM interface usage. One major difference is that COM interface usage on Windows 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 Validation

- Validates the supplied XML document against internal or external DTDs or XML Schemas.
- Checks well-formedness of XML, DTD, XML Schema, XSLT, and XQuery documents.

XSLT Transformations

- Transforms XML using supplied XSLT 1.0, 2.0, or 3.0 document.
- XML and XSLT documents can be provided as a file (via a URL) or, in the case of COM usage, as a text string.
- Output is returned as a file (at a named location) or, in the case of COM usage, as a text string.
- XSLT parameters can be supplied via the command line and via the COM interface.
- Altova extension functions, as well as XBRL, Java and .NET extension functions, enable specialized processing. This allows, for example, the creation of such features as charts and barcode in output documents.

XQuery Execution

- Executes XQuery 1.0 and 3.0 documents.
- XQuery and XML documents can be provided as a file (via a URL) or, in the case of COM usage, as a text string.
- Output is returned as a file (at a named location) or, in the case of COM usage, as a text 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.

Hyper-performance Features

- Ultra-high performance code optimizations
  - Native instruction-set implementations
  - 32-bit and 64-bit version
- Ultra-low memory footprint
  - Extremely compact in-memory representation of XML Information Set
  - Streaming instance validation
- Cross platform capabilities
- Highly scalable code for multi-CPU/multi-core/parallel computing
- Parallel loading, validation, and processing by design

Developer Features

- Superior error reporting capabilities

- Windows server mode and Unix daemon mode (via command-line options)
- Python 3.x interpreter for scripting included
- COM API on Windows platform
- Java API everywhere
- XPath Extension functions Java, .NET, XBRL, & more
- Streaming serialization
- Built-in HTTP server with REST validation API

For more information, see the section Supported Specifications and the Altova website.

1.4 Supported Specifications

RaptorXML supports the following specifications.

W3C Recommendations
Website: World Wide Web Consortium (W3C)

- Extensible Markup Language (XML) 1.0 (Fifth Edition)
- Extensible Markup Language (XML) 1.1 (Second Edition)
- Namespaces in XML 1.0 (Third Edition)
- Namespaces in XML 1.1 (Second Edition)
- XML Information Set (Second Edition)
- XML Base (Second Edition)
- XML Inclusions (XInclude) Version 1.0 (Second Edition)
- XML Linking Language (XLink) Version 1.0
- XML Schema Part 1: Structures Second Edition
- XML Schema Part 2: Datatypes Second Edition
- W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures
- W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes
- XPointer Framework
- XPointer xmlns() Scheme
- XPointer element() Scheme
- XML Path Language (XPath) Version 1.0
- XSL Transformations (XSLT) Version 1.0
- XML Path Language (XPath) 2.0 (Second Edition)
- XSL Transformations (XSLT) Version 2.0
- XQuery 1.0: An XML Query Language (Second Edition)
- XQuery 1.0 and XPath 2.0 Functions and Operators (Second Edition)
- XML Path Language (XPath) 3.0

W3C Working Drafts & Candidate Recommendations
Website: World Wide Web Consortium (W3C)

- XSL Transformations (XSLT) Version 3.0
- XQuery 3.0: An XML Query Language
- XPath and XQuery Functions and Operators 3.0

OASIS Standards
Website: OASIS Standards

- XML Catalogs V 1.1 - OASIS Standard V1.1

Chapter 2

Setting Up RaptorXML

2 Setting Up RaptorXML

This section describes procedures for correctly setting up RaptorXML Development Edition. It describes the following:

- Installation and licensing of RaptorXML on Windows systems.
- How to use XML Catalogs.
- How to work with Altova global resources.
- Security issues related to RaptorXML.

RaptorXML has special options that support XML Catalogs and Altova global resources, both of which enhance portability and modularity. You can therefore leverage the use of these features in your environment to considerable advantage.

Note: Security concerns and how to set up important security solutions are described in the section Security Issues.

2.1 On Windows

This section:

Installing on Windows
RaptorXML is available on the Altova website as a self-extracting download that will install RaptorXML with the necessary registrations. The RaptorXML executable is located by default at:

```
<ProgramFilesFolder>\Altova\RaptorXMLDevelopment2013\RaptorXMLDev.exe
```

All the necessary registrations to use RaptorXML via a COM interface, as a Java interface, and in the .NET environment will be done by the installer. This includes registering the RaptorXML executable as a COM server object, installing `RaptorXMLLib.dll` (for Java interface usage) in the `WINDIR\system32\` directory, and adding the `Altova.RaptorXML.dll` file to the .NET reference library.

Licensing on Windows
On invoking any `RaptorXMLDev` command for the first time, the license activation dialog pops up. Enter your license information in this dialog and click **Save**. If the license is valid, the dialog disappears and you can start using RaptorXML Development Edition.

Since the license of your Altova MissionKit product (XMLSpy, StyleVision, or Maporce) or Altova standalone product entitles you to use RaptorXML Development edition, entering the the details of your Altova MissionKit license or Altova standalone product in the license activation dialog will unlock RaptorXML Development Edition. No special RaptorXML Development Edition license is required.

2.2 XML Catalogs

The XML catalog mechanism enables files to be retrieved from local folders, thus increasing the overall processing speed, as well as improving the portability of documents—since only the catalog file URIs then need to be changed. See the section How Catalogs Work for details.

Altova's XML products use a catalog mechanism to quickly access and load commonly used files, such as DTDs and XML Schemas. This catalog mechanism can be customized and extended by the user, and it is described in the section Altova's XML Catalog Mechanism. The section Variables for System Locations list Windows variables for common system locations. These variables can be used in catalog files to locate commonly used folders.

This section is organized into the following sub-sections:

- How Catalogs Work
- Altova's XML Catalog Mechanism
- Variables for System Locations

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

2.2.1 How Catalogs Work

This section:

Catalogs are useful for redirecting calls to remote resources to a local URL. This is achieved by mapping, in the catalog file, public or system identifiers, URIs, or parts of identifiers or URIs to the required local URL.

Mapping public and system identifiers to local URLs
When the `DOCTYPE` declaration of a DTD in an XML file is read, the declaration's public or system identifier locates the required resource. If the identifier selects a remote resource or if the identifier is not a locator, it can still be mapped via a catalog entry to a local resource.

For example, consider the following SVG file:

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

Its public identifier is: **`-//W3C//DTD SVG 1.1//EN`**
Its system identifier is: `http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd`

A catalog entry could map the public identifier to a local URL, like this:

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

Or, a catalog entry could map the system identifier to a local URL, like this:

```
<system systemId="http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd" uri="
schemas/svg/svg11.dtd"/>
```

If there is a match for the public or system identifier in the catalog, the URL to which it is mapped is used. (Relative paths are resolved with reference to an `xml:base` attribute in the redirecting catalog element; the fallback base URL is the URL of the catalog file.) If there is no match for the public or system identifier in the catalog, then the URL in the XML document will be used (in the example above: `http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd`).

Mapping relative or absolute filepaths, Web URLs, or just names, to local URLs
The **`uri`** element can be used to map a relative or absolute filepath or a Web URL, or just any name, to a local URL, like this:

```
<uri name="doc.xslt" uri="C:\Docs\doc.xslt"/>
<uri name="U:\Docs\2013\doc.xslt" uri="C:\Docs\doc.xslt"/>
<uri name="http://www.altova.com/schemas/doc.xslt" uri="
```

```
    C:\Docs\doc.xslt"/>
• <uri name="foo" uri="C:\Docs\doc.xslt"/>
```

When the `name` value is encountered, it is mapped to the resource specified in the `uri` attribute. With a different catalog, the same name can be mapped to a different resource. For example, if you have:

```
xsi:schemaLocation="http://www.altova.com/schemas/orgchart OrgChart.xsd"
```

Normally, the URI part of the attribute's value (bold in the example above) is a path to the actual schema location. However, if the schema is referenced via a catalog, the URI part need not point to an actual XML Schema, but it does need to exist so that the lexical validity of the `xsi:schemaLocation` attribute is maintained. A value of `foo`, for example, would be sufficient for the URI part of the `xsi:schemaLocation` attribute's value (instead of `Orgchart.xsd`). The schema is located in the catalog by means of the namespace part of the `xsi:schemaLocation` attribute's value. In the example above, the namespace part is `http://www.altova.com/schemas/orgchart`.

In the catalog, the following entry would locate the schema on the basis of that namespace part.

```
<uri name="http://www.altova.com/schemas/orgchart" uri="
C:\MySchemas\OrgChart.xsd"/>
```

For more information on these elements, see the XML Catalogs specification.

2.2.2 Altova's XML Catalog Mechanism

This section:

- The root catalog file, `RootCatalog.xml`, contains the catalog files RaptorXML will look up.
- Altova's catalog extension files: `CoreCatalog.xml`, `CustomCatalog.xml`, and `Catalog.xml`.
- Supported catalog subset.

RootCatalog.xml
By default, RaptorXML will look up the file `RootCatalog.xml` (*listed below*) for the list of catalog files to use. `RootCatalog.xml` is located in the folder:

```
<ProgramFilesFolder>\Altova\RaptorXMLDevelopment2013
```

To use another file as the root catalog, use the `--catalog` option on the command line, the `setCatalog` method of the Java interface, or the `Catalog` method of the COM interface.

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog"
         xmlns:spy="http://www.altova.com/catalog_ext"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="urn:oasis:names:tc:entity:xmlns:xml:catalog
Catalog.xsd">

  <nextCatalog catalog="%PersonalFolder%/Altova/%AppAndVersionName%/
CustomCatalog.xml"/>
  <nextCatalog catalog="CoreCatalog.xml"/>

  <!-- Include all catalogs under common schemas folder on the first directory
level -->
  <nextCatalog spy:recurseFrom="%AltovaCommonFolder%/Schemas" catalog="
catalog.xml" spy:depth="1"/>

  <!-- Include all catalogs under common XBRL folder on the first directory
level -->
  <nextCatalog spy:recurseFrom="%AltovaCommonFolder%/XBRL" catalog="
catalog.xml" spy:depth="1"/>
</catalog>
```

Additional catalog files to look up are each listed in a `nextCatalog` element, and any number of these can be added. Each catalog file is looked up and the mappings in them are resolved.

In the listing above, notice that two catalogs are directly referenced: `CoreCatalog.xml` and `CustomCatalog.xml`. Additionally, catalogs named `catalog.xml` that are in the first level of subfolders of the `Schemas` and `XBRL` folders are also referenced. (The value of the `%AltovaCommonFolder%` variable is given in the section, Variables for System Locations.)

The catalog files in the Altova Common Folder map the pre-defined public and system identifiers of commonly used schemas (such as XML Schema and XHTML) to URIs that point to local copies of the respective schemas. These schemas are installed in the Altova Common Folder when RaptorXML is installed.

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

The catalog files `CoreCatalog.xml` and `CustomCatalog.xml` are listed in `RootCatalog.xml` for lookup:

- `CoreCatalog.xml` contains certain Altova-specific mappings for locating schemas in the Altova Common Folder.
- `CustomCatalog.xml` is a skeleton file in which you can create your own mappings. You can add mappings to `CustomCatalog.xml` for any schema you require but that is not addressed by the catalog files in the Altova Common Folder. Do this using the supported elements of the OASIS catalog mechanism (*see below*).
- There are a number of `Catalog.xml` files inside the folder of a specific schema or XBRL taxonomy in the Altova Common Folder, and each maps public and/or system identifiers to URIs that point to locally saved copies of the respective schemas.

Both `CoreCatalog.xml` and `CustomCatalog.xml` are in the folder, `<ProgramFilesFolder>\Altova\RaptorXMLDevelopment2013`. The `catalog.xml` files are each in a specific schema folder, these schema folders being inside the folders: `%AltovaCommonFolder%\Schemas` and `%AltovaCommonFolder%\XBRL`.

Supported catalog subset

When creating entries in a catalog file that RaptorXML will use, use only the following elements of the OASIS catalog specification. Each of the elements below is listed with an explanation of its attribute values. For a more detailed explanation, see the XML Catalogs specification.

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

In cases where there is no public identifier, the system identifier can be directly mapped to a URL via the `system` element. Also, a URI can be mapped to another URI using the `uri` element. The `rewriteURI` and `rewriteSystem` elements enable the rewriting of the starting part of a URI or system identifier, respectively. This allows the start of a filepath to be replaced and consequently enables the targeting of another directory.

Note: Each element can take the `xml:base` attribute, which is used to specify the base URI of that element. If no `xml:base` element is present, the base URI will be the URI of the catalog file.

For more information on these elements, see the XML Catalogs specification.

2.2.3 Variables for Windows System Locations

Shell environment variables can be used in catalog files to specify the path to various Windows system locations. The following variables are supported:

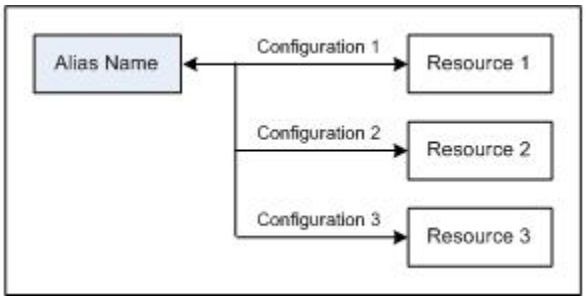
%AltovaCommonFolder%	C:\Program Files\Altova\Common2013
%DesktopFolder%	Full path to the Desktop folder for the current user.
%ProgramMenuFolder%	Full path to the Program Menu folder for the current user.
%StartMenuFolder%	Full path to Start Menu folder for the current user.
%StartUpFolder%	Full path to Start Up folder for the current user.
%TemplateFolder%	Full path to the Template folder for the current user.
%AdminToolsFolder%	Full path to the file system directory that stores administrative tools for the current user.
%AppDataFolder%	Full path to the Application Data folder for the current user.
%CommonAppDataFolder%	Full path to the file directory containing application data for all users.
%FavoritesFolder%	Full path of the Favorites folder for the current user.
%PersonalFolder%	Full path to the Personal folder for the current user.
%SendToFolder%	Full path to the SendTo folder for the current user.
%FontsFolder%	Full path to the System Fonts folder.
%ProgramFilesFolder%	Full path to the Program Files folder for the current user.
%CommonFilesFolder%	Full path to the Common Files folder for the current user.
%WindowsFolder%	Full path to the Windows folder for the current user.
%SystemFolder%	Full path to the System folder for the current user.
%LocalAppDataFolder%	Full path to the file system directory that serves as the data repository for local (non-roaming) applications.
%MyPicturesFolder%	Full path to the MyPictures folder.

2.3 Global Resources

This section:

- About global resources
- Using global resources

About global resources
An Altova global resource file maps an alias to multiple resources via different configurations, as shown in the diagram below. An alias can therefore be switched to access a different resource by switching its configuration.



Global resources are defined in Altova products, such as Altova XMLSpy, and are saved in a global resources XML file. RaptorXML is able to use global resources as inputs. To do this, it requires the name and location of the global resources file, and the alias and configuration to be used.

The advantage of using global resources is that resource can be changed merely by switching the name of teh configuration. When using RaptorXML, this means that by providing a different value of the `--globalresourcesconfig | --gc` option, a different resource can be used. (*See the example below.*)

Using global resources with RaptorXML
To specify a global resource as an input for a RaptorXML command, the following parameters are required:

- The global resources XML file (specified on the CLI with the option `--globalresourcesfile | --gr`)
- The required configuration (specified on the CLI with the option `--globalresourcesconfig | --gc`)
- The alias. This can be specified directly on the CLI where a file name is required, or it can be at a location inside an XML file where RaptorXML looks for a filename (such as in an `xsi:schemaLocation` attribute).

For example, if you wish to transform `input.xml` with `transform.xslt` to `output.html`, this would typically be achieved on the CLI with the following command that uses filenames:

```
RaptorXMLDev xslt --input=input.xml --output=output.html transform.xslt
```


If, however, you have a global resource definition that matches the alias `MyInput` to the file resource `FirstInput.xml` via a configuration called `FirstConfig`, then you could use the alias `MyInput` on the CLI as follows:

```
RaptorXMLDev xslt --input=altova://file_resource/MyInput
--gr=C:\MyGlobalResources.xml --gc=FirstConfig --output=Output.html
transform.xslt
```

Now, if you have another file resource, say `SecondInput.xml`, that is matched to the alias `MyInput` via a configuration called `SecondConfig`, then this resource can be used by changing only the `--gc` option of the previous command:

```
RaptorXMLDev xslt --input=altova://file_resource/MyInput
--gr=C:\MyGlobalResources.xml --gc=SecondConfig --output=Output.html
transform.xslt
```

Note: In the example above a file resource was used; a file resource must be prefixed with **altova://file_resource/**. You can also use global resources that are folders. To identify a folder resource, use: **altova://folder_resource/AliasName**. Note that, on the CLI, you can also use folder resources as part of a filepath. For example: **altova://folder_resource/AliasName/input.xml**.

Chapter 3

Command Line Interface (CLI)

3 Command Line Interface (CLI)

The RaptorXML executable for use with the command line interface (CLI) is located by default at:

```
<ProgramFilesFolder>\Altova\RaptorXMLDevelopment2013\bin\RaptorXMLDev.exe
```

Usage
The command line syntax is:

```
RaptorXML --h | --help | --version | <command> [options] [arguments]
```

`RaptorXML`	Calls the application.
`--h | --help`	Displays the help text.
`--version`	Displays the application's version number.
`<command>`	The command to execute. See list below. Each command is described in detail, with its options and arguments, in sub-sections of this section.
`[options]`	The options of a command. They are listed with their respective commands and are described in detail in the Options section.
`[arguments]`	The argument/s of a command. They are listed and described with their respective commands.

CLI commands
The available CLI commands are listed below, organized by functionality. They are explained in detail in the sub-sections of this section. (Note that some validation commands appear in more than one group in the list below.)

Validation commands

valdtd | dtd	Validates a DTD document.
valxml-withdtd | xml	Validates an XML document against a DTD.
valxml-withxsd | xsi	Validates an XML document against an XML Schema.
valxquery	Validates an XQuery document.
valxsd | xsd	Validates a W3C XML Schema document.
valxslt	Validates an XSLT document.
valany	Validates any document of a type validated by the preceding commands in this list. Document type is detected automatically.

Well-formedness check commands

wfxml	Checks an XML document for well-formedness.

wfdtd	Checks a DTD document for well-formedness.
wfany	Checks any XML or DTD document for well-formedness.

XSLT commands

xslt	Carries out a transformation using the XSLT file supplied by the argument.
valxslt	Validates an XSLT document.

XQuery commands

xquery	Executes an XQuery using the XQuery file supplied by the argument.
valxquery	Validates an XQuery document.

3.1 XML, DTD, XSD Validation Commands

The XML validation commands can be used to validate the following types of document:

- *XML:* Validates XML instance documents against a DTD (valxml-withdtd | xml) or an XML Schema 1.0/1.1 (valxml-withxsd | xsi).
- *DTD:* Checks that a DTD is well-formed and contains no error (valdtd | dtd).
- *XSD:* Validates a W3C XML Schema (XSD) document according to rules of the XML Schema specification (valxsd | xsd).

XML validation commands are described in detail in the sub-sections of this section:

valxml-withdtd | xml	Validates an XML instance document against a DTD.
valxml-withxsd | xsi	Validates an XML instance document against an XML Schema.
valdtd | dtd	Validates a DTD document.
valxsd | xsd	Validates a W3C XML Schema (XSD) document.
valany	Validates any one XML, DTD or XSD document. Note that this command is also used to validate XBRL (instance or taxonomy), XSLT, or XQuery, documents; the type of document submitted is detected automatically.

Note: XBRL instance, XBRL taxonomy, XSLT and XQuery documents can also be validated. These validation commands are described in their respective sections: XBRL Validation Commands, XSLT Commands, and XQuery Commands.

3.1.1 valxml-withdtd (xml)

The **valxml-withdtd | xml** command validates an XML instance document against a DTD.

```
RaptorXMLDev valxml-withdtd | xml [options] InputFile
```

The *InputFile* argument is the XML document to validate. If a reference to a DTD exists in the XML document, the `--dtd` option is not required.

Examples

- **RaptorXMLDev** valxml-withdtd --dtd=c:\MyDTD.dtd c:\Test.xml
- **RaptorXMLDev** xml c:\Test.xml
- **RaptorXMLDev** xml --verbose=true c:\Test.xml

Options
To go to an option's description, click the option or its group header.

- Boolean option values are set to `true` if the option is specified without a value.
- The values of all options can be specified without quotes except in two cases: (i) when the value string contains spaces, or (ii) when explicitly stated in the description of the option that quotes are required.

--h, --help
--version

3.1.2 valxml-withxsd (xsi)

The **valxml-withxsd | xsi** command validates an XML instance document according to the W3C XML Schema Definition Language (XSD) 1.0 and 1.1 specifications.

```
RaptorXMLDev valxml-withxsd | xsi [options] InputFile
```

The *InputFile* argument is the XML document to validate. The --schemalocation-hints=true|false indicates whether the XSD reference in the XML document is to be used or not, with the default being true (the location is used). The --xsd=*FILE* option specifies the schema/s to use.

Note: If using the --script option to run Python scripts, make sure to also specify --streaming=false.

Examples

- **RaptorXMLDev** valxml-withxsd --schemalocation-hints=false --xsd=c:\MyXSD.xsd c:\HasNoXSDRef.xml
- **RaptorXMLDev** xsi c:\HasXSDRef.xml

Options
To go to an option's description, click the option or its group header.

- Boolean option values are set to true if the option is specified without a value.
- The values of all options can be specified without quotes except in two cases: (i) when the value string contains spaces, or (ii) when explicitly stated in the description of the option that quotes are required.

Options specific to the 'valxml-withxsd' command
--assessment-mode=skip|lax|strict

XML Schema options
--schemalocation-hints=load-by-schemalocation|
load-by-namespace|
load-combining-both|
ignore
--schema-imports=load-by-schemalocation|
load-preferring-schemalocation|
load-by-namespace|
load-combining-both|
license-namespace-only
--schema-mapping=prefer-schemalocation|prefer-namespace
--xsd=*FILE*
--xsd-version=1.0|1.1|detect

XML instance options
--xinclude=true|false
--xml-mode=wf|id|valid

Processing options

3.1.3 valdtd (dtd)

The **valdtd | dtd** command validates a DTD document according to the XML 1.0 or XML 1.1 specification.

```
RaptorXMLDev valdtd | dtd [options] InputFile
```

The *InputFile* argument is the DTD document to validate.

Examples

- **RaptorXMLDev** valdtd c:\Test.dtd
- **RaptorXMLDev** dtd --verbose=true c:\Test.dtd

Options
To go to an option's description, click the option or its group header.

- Boolean option values are set to `true` if the option is specified without a value.
- The values of all options can be specified without quotes except in two cases: (i) when the value string contains spaces, or (ii) when explicitly stated in the description of the option that quotes are required.

Catalog options

--catalog=*FILE*

--user-catalog=*FILE*

Global resource options

--enable-globalresources=true|false

--gr, --globalresourcefile=*FILE*

--gc, --globalresourceconfig=*VALUE*

Error options

--error-limit=*N*|unlimited

--error-format=text|shortxml|longxml

Message options

--log-output=*FILE*

--verbose=true|false

Help and version options

--h, --help

--version

3.1.4 valxsd (xsd)

The **valxsd | xsd** command validates an XML Schema document (XSD document) according to the W3C XML Schema Definition Language (XSD) 1.0 or 1.1 specification. Note that it is the schema itself that is validated against the XML Schema specification, not an XML instance document against an XML Schema.

```
RaptorXMLDev valxsd | xsd [options] InputFile
```

The *InputFile* argument is the XML Schema document to validate. The --xsd-version=1.0|1.1|detect option specifies the XSD version to validate against, with the default being 1.0.

Examples

- **RaptorXMLDev** valxsd c:\Test.xsd
- **RaptorXMLDev** xsd --verbose=true c:\Test.xsd

Options
To go to an option's description, click the option or its group header.

- Boolean option values are set to true if the option is specified without a value.
- The values of all options can be specified without quotes except in two cases: (i) when the value string contains spaces, or (ii) when explicitly stated in the description of the option that quotes are required.

XML Schema options

--schemalocation-hints=load-by-schemalocation|
load-by-namespace|
load-combining-both|
ignore

--schema-imports=load-by-schemalocation|
load-preferring-schemalocation|
load-by-namespace|
load-combining-both|
license-namespace-only

--schema-mapping=prefer-schemalocation|prefer-namespace

--xsd-version=1.0|1.1|detect

Processing options

--script=*FILE*

Catalog options

--catalog=*FILE*

--user-catalog=*FILE*

Global resource options

--enable-globalresources=true|false

--gr, --globalresourcefile=*FILE*

--gc, --globalresourceconfig=*VALUE*

XML instance options

--xinclude=true|false
--xml-mode=wf|id|valid

Error options

--error-limit=*N*|unlimited
--error-format=text|shortxml|longxml

Message options

--log-output=*FILE*
--verbose=true|false

Help and version options

--h, --help
--version

3.1.5 valany

The **valany** command validates an XML, DTD, or XML Schema document according to the respective specification/s. The type of document is detected automatically.

```
RaptorXMLDev valany [options] InputFile
```

The *InputFile* argument is the document to validate. Note that only one document can be submitted as the argument of the command. The type of the submitted document is detected automatically.

Examples

- **RaptorXMLDev** valany c:\Test.xml
- **RaptorXMLDev** valany --errorformat=text c:\Test.xml

Options
To go to an option's description, click the option or its group header.

- Boolean option values are set to `true` if the option is specified without a value.
- The values of all options can be specified without quotes except in two cases: (i) when the value string contains spaces, or (ii) when explicitly stated in the description of the option that quotes are required.

XML Schema options

--schemalocation-hints=load-by-schemalocation|
load-by-namespace|
load-combining-both|
ignore

--schema-imports=load-by-schemalocation|
load-preferring-schemalocation|
load-by-namespace|
load-combining-both|
license-namespace-only

--schema-mapping=prefer-schemalocation|prefer-namespace

Catalog options

--catalog=*FILE*

--user-catalog=*FILE*

Global resource options

--enable-globalresources=true|false

--gr, --globalresourcefile=*FILE*

--gc, --globalresourceconfig=*VALUE*

Error options

--error-limit=*N*|unlimited

--error-format=text|shortxml|longxml

Message options

--log-output=*FILE*
--verbose=true|false

Help and version options

--h, --help
--version

3.2 Well-formedness Check Commands

The well-formedness check commands can be used to check the well-formedness of XML documents and DTDs. These commands are listed below and described in detail in the sub-sections of this section:

wfxml	Checks the well-formedness of XML documents.
wfdtd	Checks the well-formedness of DTDs.
wfany	Checks the well-formedness of an XML document or DTD. Type is detected automatically.

3.2.1 wfxml

The **wfxml** command checks an XML document for well-formedness according to the XML 1.0 or XML 1.1 specification.

```
RaptorXMLDev wfxml [options] InputFile
```

The *InputFile* argument is the XML document to check for well-formedness.

Examples

- **RaptorXMLDev** wfxml c:\Test.xml
- **RaptorXMLDev** wfxml --verbose=true c:\Test.xml

Options
To go to an option's description, click the option or its group header.

- Boolean option values are set to `true` if the option is specified without a value.
- The values of all options can be specified without quotes except in two cases: (i) when the value string contains spaces, or (ii) when explicitly stated in the description of the option that quotes are required.

3.2.2 wfdtd

The **wfdtd** command checks a DTD document for well-formedness according to the XML 1.0 or XML 1.1 specification.

```
RaptorXMLDev wfdtd [options] InputFile
```

The *InputFile* argument is the DTD document to check for well-formedness.

Examples

- **RaptorXMLDev** wfdtd c:\Test.dtd
- **RaptorXMLDev** wfdtd --verbose=true c:\Test.dtd

Options
To go to an option's description, click the option or its group header.

- Boolean option values are set to `true` if the option is specified without a value.
- The values of all options can be specified without quotes except in two cases: (i) when the value string contains spaces, or (ii) when explicitly stated in the description of the option that quotes are required.

Catalog options
--catalog=*FILE*
--user-catalog=*FILE*

Global resource options
--enable-globalresources=true|false
--gr, --globalresourcefile=*FILE*
--gc, --globalresourceconfig=*VALUE*

Error options
--error-limit=*N*|unlimited
--error-format=text|shortxml|longxml

Message options
--log-output=*FILE*
--verbose=true|false

Help and version options
--h, --help
--version

3.2.3 wfany

The **wfany** command checks an XML or DTD document for well-formedness according to the respective specification/s. The type of document is detected automatically.

```
RaptorXMLDev wfany [options] InputFile
```

The *InputFile* argument is the document to check for well-formedness. Note that only one document can be submitted as the argument of the command. The type of the submitted document is detected automatically.

Examples

- **RaptorXMLDev** wfany c:\Test.xml
- **RaptorXMLDev** wfany --errorformat=text c:\Test.xml

Options
To go to an option's description, click the option or its group header.

- Boolean option values are set to `true` if the option is specified without a value.
- The values of all options can be specified without quotes except in two cases: (i) when the value string contains spaces, or (ii) when explicitly stated in the description of the option that quotes are required.

Catalog options

--catalog=*FILE*

--user-catalog=*FILE*

Global resource options

--enable-globalresources=true|false

--gr, --globalresourcefile=*FILE*

--gc, --globalresourceconfig=*VALUE*

Error options

--error-limit=*N*|unlimited

--error-format=text|shortxml|longxml

Message options

--log-output=*FILE*

--verbose=true|false

Help and version options

--h, --help

--version

3.3 XSLT Commands

The XSLT commands are:

- xslt: for transforming XML documents with an XSLT document
- valxslt: for validating XSLT documents

The arguments and options for each command are listed in the sub-sections, xslt and valxslt.

3.3.1 xslt

The **xslt** command takes an XSLT file as its single argument and uses it to transform an input XML file to produce an output file. The input and output files are specified as options.

```
RaptorXMLDev xslt [options] XSLT-File
```

The *XSLT-File* argument is the path and name of the XSLT file to use for the transformation. An input XML file (--input) or a named template entry point (--template-entry-point) is required. If no --output option is specified, output is written to standard output. You can use XSLT 1.0, 2.0, or 3.0. By default XSLT 3.0 is used.

Examples

- **RaptorXMLDev** xslt --input=c:\Test.xml --output=c:\Output.xml c:\Test.xslt
- **RaptorXMLDev** xslt --template-entry-point=StartTemplate --output=c:\Output.xml c:\Test.xslt
- **RaptorXMLDev** xslt --input=c:\Test.xml --output=c:\Output.xml --param date=//node/@att1 --p=title:'stringwithoutspace' --param=title:"'string with spaces'" --p=amount:456 c:\Test.xslt

---

Options
To go to an option's description, click the option or its group header.

- Boolean option values are set to `true` if the option is specified without a value.
- The values of all options can be specified without quotes except in two cases: (i) when the value string contains spaces, or (ii) when explicitly stated in the description of the option that quotes are required.

Options specific to the 'xslt' command

--indent-characters=*VALUE*
--input=*FILE*
--output=*FILE*
--p, --param=*KEY:VALUE*
--streaming-serialization-enabled=true|false

Options common to the 'xslt' and 'valxslt' commands

--chartext-disable=true|false
--dotnetext-disable=true|false
--javaext-barcode-location=*FILE*
--javaext-disable=true|false
--template-entry-point=*VALUE*
--template-mode=*VALUE*
--xslt-version=1|2|3

Catalog options

--catalog=*FILE*

--user-catalog=*FILE*

Global resource options

--enable-globalresources=true|false

--gr, --globalresourcefile=*FILE*

--gc, --globalresourceconfig=*VALUE*

Error options

--error-limit=*N*|unlimited

Message options

--verbose=true|false

XML instance options

--xinclude=true|false

--xml-mode=wf|id|valid

XML Schema options

--schemalocation-hints=load-by-schemalocation|
load-by-namespace|
load-combining-both|
ignore

--schema-imports=load-by-schemalocation|
load-preferring-schemalocation|
load-by-namespace|
load-combining-both|
license-namespace-only

--schema-mapping=prefer-schemalocation|prefer-namespace

--xsd-version=1.0|1.1|detect

Help and version options

--h, --help

--version

3.3.2 valxslt

The **valxslt** command takes an XSLT file as its single argument and validates it.

```
RaptorXMLDev valxslt [options] XSLT-File
```

The ***XSLT-File*** argument is the path and name of the XSLT file to be validated. Validation can be according to the XSLT 1.0, 2.0, or 3.0 specification. By default XSLT 3.0 is the specification used.

Examples

- **RaptorXMLDev** valxslt c:\Test.xslt
- **RaptorXMLDev** valxslt --xslt-version=2 c:\Test.xslt

Options
To go to an option's description, click the option or its group header.

- Boolean option values are set to `true` if the option is specified without a value.
- The values of all options can be specified without quotes except in two cases: (i) when the value string contains spaces, or (ii) when explicitly stated in the description of the option that quotes are required.

--xinclude=true|false

--xml-mode=wf|id|valid

XML Schema options

--schemalocation-hints=load-by-schemalocation|
load-by-namespace|
load-combining-both|
ignore

--schema-imports=load-by-schemalocation|
load-preferring-schemalocation|
load-by-namespace|
load-combining-both|
license-namespace-only

--schema-mapping=prefer-schemalocation|prefer-namespace

--xsd-version=1.0|1.1|detect

Help and version options

--h, --help

--version

3.4 XQuery Commands

The XQuery commands are:

- xquery: for executing XQuery documents, optionally with an input document
- valxquery: for validating XQuery documents

The arguments and options for each command are listed in the sub-sections, xquery and valxquery.

3.4.1 xquery

The **xquery** command takes an XQuery file as its single argument and executes it with an optional input file to produce an output file. The input and output files are specified as options.

```
RaptorXMLDev xquery [options] XQuery-File
```

The argument ***XQuery-File*** is the path and name of the XQuery file to be executed.

Examples

- **RaptorXMLDev** xquery --output=c:\Output.xml c:\TestQuery.xq
- **RaptorXMLDev** xquery --input=c:\Input.xml --output=c:\Output.xml --var company=Altova -var date=2006-01-01 c:\TestQuery.xq
- **RaptorXMLDev** xquery --input=c:\Input.xml --output=c:\Output.xml --xparam source=" doc('c:\test\books.xml')//book "
- **RaptorXMLDev** xquery --output=c:\Output.xml --omit-xml-declaration=false --output-encoding=ASCII c:\TestQuery.xq

---

Options
To go to an option's description, click the option or its group header.

- Boolean option values are set to `true` if the option is specified without a value.
- The values of all options can be specified without quotes except in two cases: (i) when the value string contains spaces, or (ii) when explicitly stated in the description of the option that quotes are required.

Options specific to the 'xquery' command

--indent-characters=*VALUE*
--input=*FILE*
--output=*FILE*
--output-encoding=*VALUE*
--output-indent=true|false
--output-method=xml|html|xhtml|text
--p, --param=*KEY:VALUE*

Options common to the 'xquery' and 'valxquery' commands

--omit-xml-declaration=true|false
--xquery-version=1|3

Catalog options

--catalog=*FILE*
--user-catalog=*FILE*

Global resource options

--enable-globalresources=true|false
--gr, --globalresourcefile=*FILE*
--gc, --globalresourceconfig=*VALUE*

Error options

--error-limit=*N*|unlimited

Message options

--verbose=true|false

XML instance options

--xinclude=true|false

--xml-mode=wf|id|valid

XMLSchema options

--xsd-version=1.0|1.1|detect

Help and version options

--h, --help

--version

3.4.2 valxquery

The **valxquery** command takes an XQuery file as its single argument and validates it.

```
RaptorXMLDev valxquery [options] XQuery-File
```

The *XQuery-File* arguments is the path and name of the XQuery file to be validated.

Examples

- **RaptorXMLDev** valxquery c:\Test.xquery
- **RaptorXMLDev** valxquery --xquery-version=1 c:\Test.xquery

---

Options
To go to an option's description, click the option or its group header.

- Boolean option values are set to `true` if the option is specified without a value.
- The values of all options can be specified without quotes except in two cases: (i) when the value string contains spaces, or (ii) when explicitly stated in the description of the option that quotes are required.

--version

3.5 Help and License Commands

This section describes two important features of RaptorXML:

- Help Command: Describes how to display information about available commands, or about a command's arguments and options
- License Commands: Describes how top license RaptorXML

3.5.1 Help Command

The **help** command takes a single argument: the name of the command for which help is required. It displays the syntax of the command and other information relevant to the correct execution of the command.

```
RaptorXMLDev help Command
```

Note: When no argument is submitted, running the help command causes all available commands to be displayed, each with a short description of what it does.

Example
Example of the help command:

```
RaptorXML help valany
```

The command above contains one argument: the command valany, for which help is required. When this command is executed, it will display help information about the valany command.

The --help option
Help information about a command is also available by using the --help option with that command. For example, using the --help option with the valany command, as follows:

```
RaptorXML valany --help
```

achieves the same result as does using the help command with an argument of valany:

```
RaptorXML help valany
```

In both cases, help information about the valany command is displayed.

3.5.2 License Commands

On invoking any `RaptorXMLDev` command for the first time, the license activation dialog pops up. Enter your license information in this dialog and click **Save**. If the license is valid, the dialog disappears and you can start using RaptorXML Development Edition.

Note about licenses
Since the license of your Altova MissionKit product or Altova standalone product entitles you to use RaptorXML Development edition, the details of your Altova MissionKit license or Altova standalone product will unlock RaptorXML Development Edition. No special RaptorXML Development Edition license is required.

3.6 Options

This section contains a description of all CLI options, organized by functionality. To find out which options may be used with each command, see the description of the respective commands.

- Catalogs
- Errors
- Global Resources
- Help and Version
- Messages
- Processing
- XBRL Evaluation
- XBRL Schemas
- XML Document
- XML Validation
- XML Schema Document (XSD)
- XQuery
- XSLT
- ZIP Files

3.6.1 Catalogs

[--catalog, --user-catalog]

--catalog=*FILE*
Specifies the absolute path to a root catalog file that is not the installed root catalog file. The default value is the absolute path to the installed root catalog file.

--user-catalog=*FILE*
Specifies the absolute path to an XML catalog to be used in addition to the root catalog.

Note: Boolean option values are set to `true` if the option is specified without a value.

3.6.2 Errors

[--error-limit, --error-format]

`--error-limit=N|unlimited`
Specifies the error limit. Default value is `100`. Useful for limiting processor use during validation. When the error limit is reached, validation stops.

`--error-format=text|shortxml|longxml`
Specifies the format of the error output. Default value is `text`. The other options generate XML formats, with `longxml` generating more detail.

Note: Boolean option values are set to `true` if the option is specified without a value.

3.6.3 Global Resources

[--enable-global-resources, --globalresourcefile, --globalresourceconfig]

--enable-globalresources=true|false
Enables global resources. Default value is `false`.

--gr, --globalresourcefile=*FILE*
Specifies the global resource file (and enables global resources).

--gc, --globalresourceconfig=*VALUE*
Specifies the active configuration of the global resource (and enables global resources).

Note: Boolean option values are set to `true` if the option is specified without a value.

3.6.4 Help and Version

[--help, --version]

--h, --help
Displays help text for the command. For example, `valany --h`. (Alternatively the `help` command can be used with an argument. For example: `help valany`.)

--version
Displays the version of RaptorXML. If used with a command, place `--version` before the command.

Note: Boolean option values are set to `true` if the option is specified without a value.

3.6.5 Messages

[--log-output, --verbose]

--log-output=*FILE*
Writes the message output to the specified file URL instead of to the console. Ensure that the CLI has write permission to the output location.

--verbose=true|false
A value of `true` enables output of additional information during validation. Default value is `false`.

Note: Boolean option values are set to `true` if the option is specified without a value.

3.6.6 Processing

[--listfile, --script, --streaming-serialization-enabled, --streaming]

--listfile=true|false
If `true`, treats the command's *`InputFile`* argument as a text file containing one filename per line. Default value is `false`. (An alternative is to list the files on the CLI with a space as separator. Note, however, that CLIs have a maximum-character limitation.) Note that the `--listfile` option applies only to arguments, and not to options.

--script=*File*
Executes the Python script in the submitted file after validation has been completed.

--streaming=true|false
Enables streaming validation. Default is `true`. In streaming mode, data stored in memory is minimized and processing is faster. The downside is that information that might be required subsequently—for example, a data model of the XML instance document—will not be available. In situations where this is significant, streaming mode will need to be turned off (by giving `--streaming` a value of `false`). When using the `--script` option with the `valxml-withxsd` command, disable streaming.

Note: Boolean option values are set to `true` if the option is specified without a value.

3.6.7 XML Instance

[--xinclude, --xml-mode]

--xinclude=true|false
Enables XML Inclusions (XInclude) support. Default value is `false`. When `false`, XInclude's `include` elements are ignored.

--xml-mode=wf|id|valid
Specifies the XML processing mode to use: `wf`=wellformed check; `id`=wellformed with ID/IDREF checks; `valid`=validation. Default value is `wf`.

Note: Boolean option values are set to `true` if the option is specified without a value.

3.6.8 XML Instance Validation

[--dtd, --xsd, --namespaces, --assessment-mode]

--dtd=*FILE*
Specifies the external DTD document to use for validation. If a reference to an external DTD is present in the XML document, then the CLI option overrides the external reference.

--xsd=*FILE*
Specifies one or more XML Schema documents to use for the validation of XML instance documents. Add the option multiple times to specify multiple schema documents.

--namespaces=true|false
Enables namespace-aware processing. This is useful for checking the XML instance for errors due to incorrect namespaces. Default value is `false`.

--assessment-mode=skip|lax|strict
Specifies the schema-validity assessment mode as defined in the XSD specifications. Default value is `strict`. The XML instance document will be validated according to the mode specified with this option.

Note: Boolean option values are set to `true` if the option is specified without a value.

3.6.9 XML Schema Document (XSD)

[--schemalocation-hints, --schema-imports, --schema-mapping, --xsd-version]
[Note about schema location hints]

--schemalocation-hints=load-by-schemalocation|
load-by-namespace|
load-combining-both|
ignore

- The `load-by-schemalocation` value uses the URL of the schema location in the `xsi:schemaLocation` and `xsi:noNamespaceSchemaLocation` attributes in XML instance documents. This is the **default value**.
- The `load-by-namespace` value takes the namespace part of `xsi:schemaLocation` and an empty string in the case of `xsi:noNamespaceSchemaLocation` and locates the schema via a catalog mapping.
- If `load-combining-both` is used and if either the namespace part or the URL part has a catalog mapping, then the catalog mapping is used. If both have catalog mappings, then the value of the `--schema-mapping` option decides which mapping is used. If neither the namespace nor URL has a catalog mapping, the URL is used.
- If the option's value is `ignore`, then the `xsi:schemaLocation` and `xsi:noNamespaceSchemaLocation` attributes are both ignored.

--schema-imports=load-by-schemalocation|
load-preferring-schemalocation|
load-by-namespace|
load-combining-both|
license-namespace-only

Specifies the behaviour of `xs:import` elements, each of which has an optional `namespace` attribute and an optional `schemaLocation` attribute: `<import namespace="someNS" schemaLocation="someURL">`. The behavior is as follows:

- `load-by-schemalocation`: The value of the `schemaLocation` attribute is used to locate the schema, taking account of catalog mappings. If the namespace attribute is present, the namespace is imported (licensed).
- `load-preferring-schemalocation`: If the `schemaLocation` attribute is present, it is used, taking account of catalog mappings. If no `schemaLocation` attribute is present, then the value of the `namespace` attribute is used via a catalog mapping. This is the **default value**.
- `load-by-namespace`: The value of the `namespace` attribute is used to locate the schema via a catalog mapping.
- `load-combining-both`: If either the `namespace` or `schemaLocation` attribute has a catalog mapping, then the mapping is used. If both have catalog mappings, then the value of the `--schema-mapping` option decides which mapping is used. If no catalog mapping is present, the `schemaLocation` attribute is used.
- `license-namespace-only`: The namespace is imported. No schema document is imported.

--schema-mapping=prefer-schemalocation|prefer-namespace
If either the --schemalocation-hints or the --schema-imports option has a value of load-combining-both, and if the namespace and URL parts involved both have catalog mappings, then the value of this option specifies which of the two mappings to use (namespace mapping or URL mapping; the prefer-schemalocation value refers to the URL mapping). Default is prefer-schemalocation.

--xsd-version=1.0|1.1|detect
Specifies the W3C Schema Definition Language (XSD) version to use. Default is 1.0. This option can also be useful to find out in what ways a schema which is 1.0-compatible is not 1.1-compatible. The detect option is an Altova-specific feature. It enables the version of the XML Schema document (1.0 or 1.1) to be detected by reading the value of the version attribute of the document's <xs:schema> element. If the value of the @version attribute is 1.1, the schema is detected as being version 1.1. For any other value, the schema is detected as being version 1.0. Note that this latter mechanism is not defined in the XML Schema specification; it is intended to conform with the schema-detection mechanism used in other Altova applications.

Note about schema location hints
Instance documents can use hints to indicate the schema location. Two attributes are used for hints:

- xsi:schemaLocation for schema documents with target namespaces. The attribute's value is a pair of items, the first of which is a namespace, the second is a URL that locates a schema document. The namespace name must match the target namespace of the schema document.

```
<document xmlns="http://www.altova.com/schemas/test03"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://www.altova.com/schemas/test03
Test.xsd">
```

- xsi:noNamespaceSchemaLocation for schema documents without target namespaces. The attribute's value is the schema document's URL. The referenced schema document must have no target namespace.

```
<document xmlns="http://www.altova.com/schemas/test03"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="Test.xsd">
```

The --schemalocation-hints option specifies how these two attributes are to be used as hints, especially how the schemaLocation attribute information is to be handled (*see the option's description above*). Note that RaptorXML Development Edition considers the namespace part of the xsi:noNamespaceSchemaLocation value to be the empty string.

Schema location hints can also be given in an import statement of an XML Schema document.

```
<import namespace="someNS" schemaLocation="someURL">
```

In the `import` statement, too, hints can be given via a namespace that can be mapped to a schema in a catalog file, or directly as a URL in the `schemaLocation` attribute. The `--schema-imports` option specifies how the schema location is to be selected.

3.6.10 XQuery

Options specific to the `xquery` command
[--indent-characters, --input, --output, --output-encoding, --output-indent, --output-method, --param]

`--indent-characters=`*`VALUE`*
Specifies the character string to be used as indentation.

`--input=`*`FILE`*
The URL of the XML file to be transformed.

`--output=`*`FILE`*
The URL of the primary-output file. For example, in the case of multiple-file HTML output, the primary-output file will be the location of the entry point HTML file. If no `--output` option is specified, output is written to standard output.

`--output-encoding=`*`VALUE`*
The value of the encoding attribute in the output document. Valid values are names in the IANA character set registry. Default value is `UTF-8`.

`--output-indent=true|false`
If `true`, the output will be indented according to its hierarchic structure. If `false`, there will be no hierarchical indentation. Default is `false`.

`--output-method=xml|html|xhtml|text`
Specifies the output format. Default value is `xml`.

`--p, --param=`*`KEY:VALUE`*
Specifies the value of an external parameter. An external parameter is declared in the XQuery document with the `declare variable` declaration followed by a variable name and then the `external` keyword followed by the trailing semi-colon. For example:

```
declare variable $foo as xs:string external;
```

Because of the `external` keyword `$foo` becomes an external parameter, the value of which is passed at runtime from an external source. The external parameter is given a value with the CLI command. For example:

```
--param=foo:'MyName'
```

In the description statement above, *`KEY`* is the external parameter name, *`VALUE`* is the value of the external parameter, given as an XPath expression. Parameter names used on the CLI must be declared in the XQuery document. If multiple external parameters are passed values on the CLI, each must be given a separate `--param` option. Double quotes must be used if the XPath expression contains spaces.

Note: Boolean option values are set to `true` if the option is specified without a value.

Options common to the `xquery` and `valxquery` commands
[--omit-xml-declaration, --xquery-version]

`--omit-xml-declaration=true|false`
Serialization option to specify whether the XML declaration should be omitted from the output or not. If `true`, there will be no XML declaration in the output document. If `false`, an XML declaration will be included. Default value is `false`.

--xquery-version=1|3
Specifies whether the XQuery processor should use XQuery 1.0 or XQuery 3.0. Default value is `3`.

Note: Boolean option values are set to `true` if the option is specified without a value.

3.6.11 XSLT

Options specific to the `xslt` command
[--indent-characters, --input, --output, --param, --streaming-serialization-enabled]

`--indent-characters=`*`VALUE`*
Specifies the character string to be used as indentation.

`--input=`*`FILE`*
The URL of the XML file to be transformed.

`--output=`*`FILE`*
The URL of the primary-output file. For example, in the case of multiple-file HTML output, the primary-output file will be the location of the entry point HTML file. If no `--output` option is specified, output is written to standard output.

`--p, --param=`*`KEY:VALUE`*
Specifies a global stylesheet parameter. *`KEY`* is the parameter name, *`VALUE`* is an XPath expression that provides the parameter value. Parameter names used on the CLI must be declared in the stylesheet. If multiple parameters are used, the `--param` switch must be used before each parameter. Double quotes must be used around the XPath expression if it contains a space—whether the space is in the XPath expression itself or in a string literal in the expression. *For example:*

```
RaptorXMLDev xslt --input=c:\Test.xml --output=c:\Output.xml --param
date=//node/@att1 --p=title:'stringwithoutspace' --param=title:"'string with
spaces'" --p=amount:456 c:\Test.xslt
```

`--streaming-serialization-enabled=true|false`
Enables streaming serialization. Default value is `true`.

Note: Boolean option values are set to `true` if the option is specified without a value.

Options common to the `xslt` and `valxslt` commands
[--chartext-disable, --dotnetext-disable, --javaext-barcode-location, --javaext-disable,, --template-entry-point, --template-mode, --xslt-version]

`--chartext-disable=true|false`
Disables chart extensions. Default value is `false`. *Not available in Development Edition.*

`--dotnetext-disable=true|false`
Disables .NET extensions. Default value is `false`.

`--javaext-barcode-location=`*`FILE`*
Specifies the location of the barcode extension file.

`--javaext-disable=true|false`
Disables Java extensions. Default value is `false`.

`--template-entry-point=`*`VALUE`*
Gives the name of a named template in the XSLT stylesheet that is the entry point of the transformation.

--template-mode=*VALUE*
Specifies the template mode to use for the transformation.

--xslt-version=1|2|3
Specifies whether the XSLT processor should use XSLT 1.0, XSLT 2.0, or XSLT 3.0. Default value is `3`.

Note: Boolean option values are set to `true` if the option is specified without a value.

3.6.12 ZIP Files

[--recurse]

--recurse=true|false
Used to select files within a ZIP archive. If `true`, the command's *`InputFile`* argument will select the specified file also in subdirectories. For example: `test.zip|zip\test.xml` will select files named `test.xml` at all folder levels of the zip folder. The wildcard characters `*` and `?` may be used. So, `*.xml` will select all `.xml` files in the zip folder. The parameter's default value is `false`. It is not available in the Development Edition.

Note: Boolean option values are set to `true` if the option is specified without a value.

Chapter 4

Java Interface

4 Java Interface

The Java API is packaged in the `com.altova.raptorxml` package. The entry point is the RaptorXMLFactory interface, which provides methods for getting engine objects for validation and transformation.

The getter methods always returns new objects. A RaptorXMLFactory instance can be created by calling RaptorXML.getFactory().

[This factory method returns different factories for the RaptorXMLServer product and for the RaptorXMLDevDevelopment Edition product).

The RaptorXMLFactory's public interface is described by the following code:

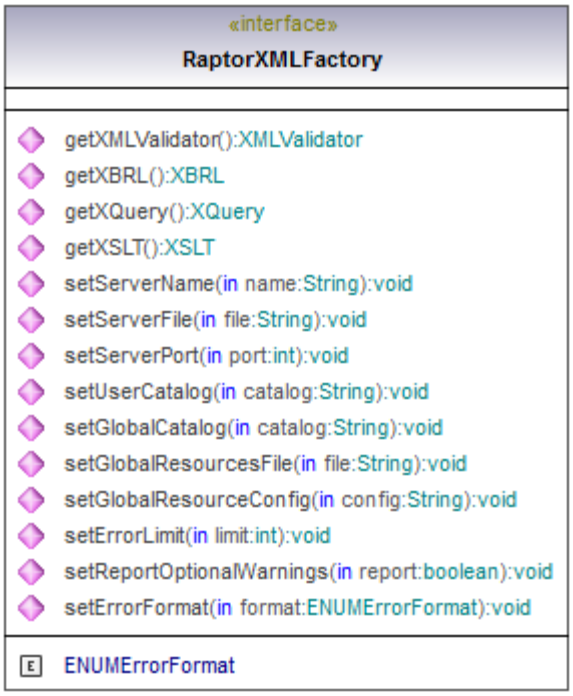
```
public interface RaptorXMLFactory
{
   public XMLValidator getXMLValidator();
   public XSLT getXBRL();
   public XQuery getXQuery();
   public XSLT getXSLT();
   public void setServerName( String name ) throws RaptorXMLException;
   public void setServerFile( String file ) throws RaptorXMLException;
   public void setServerPort( int port ) throws RaptorXMLException;
     public void setUserCatalog(String catalog);
     public void setGlobalResourcesFile(String file);
     public void setGlobalResourceConfig(String config);
     public void setErrorLimit(int limit);
     public void setReportOptionalWarnings(boolean report);
```

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

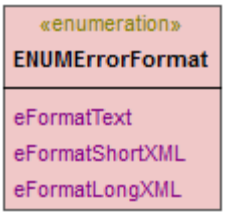
- RaptorXMLFactory
  Creates new RaptorXML COM server object instance via native call, and provides access to RaptorXML engines.
- XMLValidator
  Class holding XMLValidator.
- XBRL
  Class holding the XBRL Validator
- XSLT
  Class holding the XSLT 1.0, 2.0, 3.0 Engines.
- XQuery
  Class holding the XQuery 1.0, 3.0 Engines.

4.1 RaptorXMLJava - RaptorXMLFactory

Interface **RaptorXMLFactory**

diagram	
documentation	Use RaptorXMLFactory() to create a new RaptorXML COM server object instance. This provides access to the RaptorXML engine. The relationship between RaptorXMLFactory and the RaptorXML COM object is one-to-one.

Enumeration **RaptorXMLFactory::ENUMErrorFormat**

diagram	
typedElements	Interface **RaptorXMLFactory** Operation **setErrorFormat**
documentation	Contains the enumeration literals defining the format of the error output.

EnumerationLiteral **RaptorXMLFactory::ENUMErrorFormat::eFormatLongXML**

documentation	Sets the error output format to Long XML. This XML output format provides the most detail of the output formats.

EnumerationLiteral **RaptorXMLFactory::ENUMErrorFormat::eFormatShortXML**

documentation	Sets the error output format to Short XML. This XML output format is an abbreviated form of the XML Long format.

EnumerationLiteral **RaptorXMLFactory::ENUMErrorFormat::eFormatText**

documentation	Sets the error output format to "Text". This is the default setting.

Operation **RaptorXMLFactory::getXBRL**

parameter	name	direction	type	multiplicity	default
	return	**return**	**XBRL**		
documentation	Retrieves the XBRL engine and returns a new XBRL instance of RaptorXMLFactory. Only supported by RaptorXML+XBRL Server. Parameters: none				

Operation **RaptorXMLFactory::getXMLValidator**

parameter	name	direction	type	multiplicity	default
	return	**return**	**XMLValidator**		
documentation	Retrieves the XML engine and returns a new XML validator instance of RaptorXMLFactory. Parameters: none				

Operation **RaptorXMLFactory::getXQuery**

parameter	name	direction	type	multiplicity	default
	return	**return**	**XQuery**		
documentation	Retrieves the XQuery engine and returns a new XQuery instance of RaptorXMLFactory. Parameters: none				

Operation **RaptorXMLFactory::getXSLT**

parameter	name	direction	type	multiplicity	default
	return	**return**	**XSLT**		
documentation	Retrieves the XSLT engine and returns a new XSLT instance of XMLFactory. Parameters: none				

Operation **RaptorXMLFactory::setErrorFormat**

parameter	name	direction	type	multiplicity	default
	format	**in**	**ENUMErrorFormat**		
	return	**return**	**void**		
documentation	Sets the RaptorXML error format (Text, ShortXML, LongXML), depends on the ENUMErrorFormat literals. Parameters: 'format' holds the value of the selected enumeration literal, - of type ENUMErrorFormat.				

Operation **RaptorXMLFactory::setErrorLimit**

parameter	name	direction	type	multiplicity	default
	limit	**in**	**int**		
	return	**return**	**void**		
documentation	Configures the RaptorXML validation error limit property. Properties: 'limit' defines the number of errors to be reported before halting execution - of type int. Use -1 to define an unlimited number of errors, i.e. all validation errors will be reported.				

Operation **RaptorXMLFactory::setGlobalCatalog**

parameter	name	direction	type	multiplicity	default
	catalog	**in**	**String**		
	return	**return**	**void**		
documentation	Sets the file name (as a URL) of the global catalog (e.g. RootCatalog.xml). Parameters: 'catalog" is the URL for the base location of the global catalog file - of type string. Note: Use the absolute URL for the input file name. An absolute URL specifies the exact location of the file, e.g. http://www.myWebsite.com/xmlfiles/myGlobCatalog.xml				

Operation **RaptorXMLFactory::setGlobalResourceConfig**

parameter	name	direction	type	multiplicity	default
	config	**in**	**String**		
	return	**return**	**void**		
documentation	Sets the active configuration of the global resource. Parameters: 'config' is the name of the configuration used by the active global resource - of type string.				

Operation **RaptorXMLFactory::setGlobalResourcesFile**

parameter	name	direction	type	multiplicity	default
	file	**in**	**String**		
	return	**return**	**void**		
documentation	Sets the file name/URL for the global resource XML file (e.g. GlobalResources.xml). Parameters: 'file' holds the name of the XML file - of type string. Note: Use the absolute URL for the input file name. An absolute URL specifies the exact location of the file, e.g. http://www.myWebsite.com/xmlfiles/myInputxml.xml				

Operation **RaptorXMLFactory::setReportOptionalWarnings**

parameter	name	direction	type	multiplicity	default
	report	**in**	**boolean**		
	return	**return**	**void**		
documentation	Enables / disables the reporting of warnings. Parameters: 'report' holds the boolean value - of type boolean. Values: 'true' enables reporting, 'false' disables it.				

Operation **RaptorXMLFactory::setServerFile**

parameter	name	direction	type	multiplicity	default
	file	**in**	**String**		
	return	**return**	**void**		
documentation	Sets the server address for the HTTP server (this method fails in RaptorXML Development Edition). Parameters: 'file' is the server address (relative to the HTTP server's root) - of type string. Raises a RaptorXMLException when an error occurs.				

Operation **RaptorXMLFactory::setServerName**

parameter	name	direction	type	multiplicity	default
	name	**in**	**String**		
	return	**return**	**void**		
documentation	Sets the server name for the HTTP server (this method fails in RaptorXML Development Edition). Parameters: 'name' is the server name - of type string. Raises a RaptorXMLException when an error occurs.				

Operation **RaptorXMLFactory::setServerPort**

parameter	name	direction	type	multiplicity	default
	port	**in**	**int**		
	return	**return**	**void**		
documentation	Sets the server port for the HTTP server (this method fails in RaptorXML Development Edition). Parameters: 'port' is the server port - of type int. Raises a RaptorXMLException when an error occurs.				

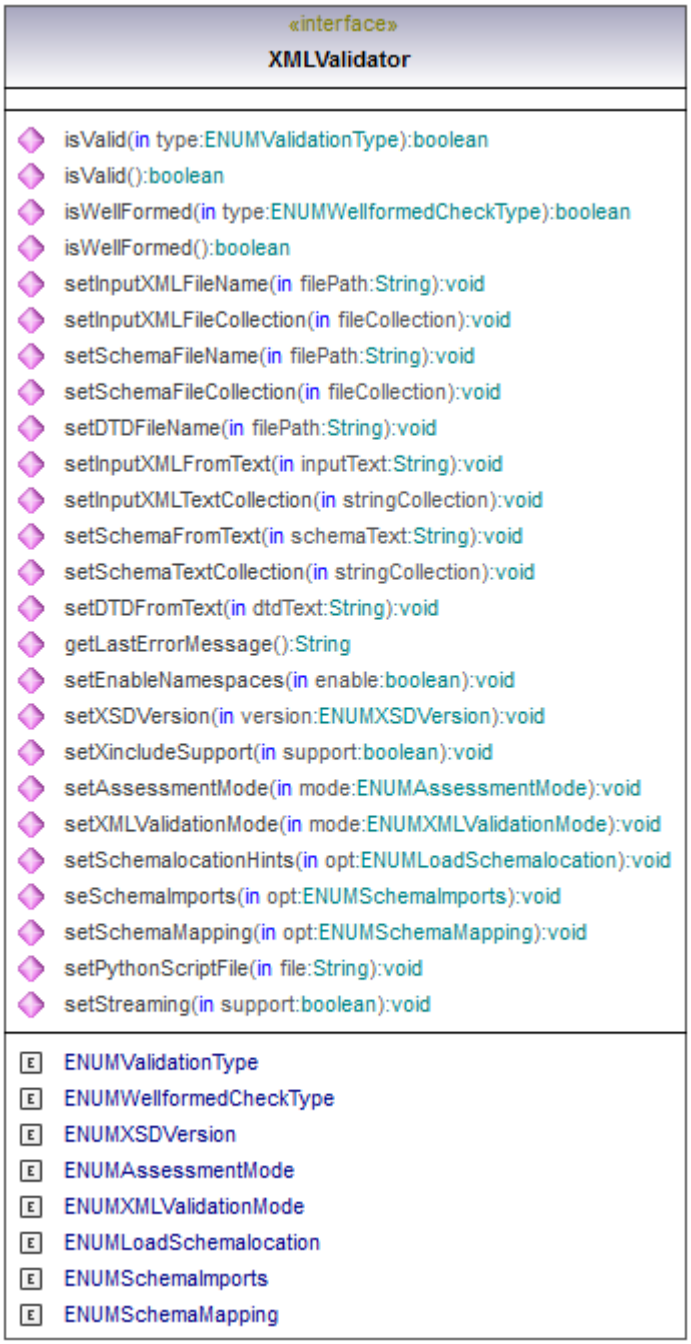
Operation **RaptorXMLFactory::setUserCatalog**

parameter	name	direction	type	multiplicity	default
	catalog	**in**	**String**		
	return	**return**	**void**		
documentation	Sets the name (as a URL) for the user-defined catalog (e.g. CustomCatalog.xml). Parameters: 'catalog' is the name of the user-defined catalog - of type string. Note: Use the absolute URL for the input file name. An absolute URL specifies the exact location of the file.				

4.2 RaptorXMLJava - XMLValidator

Interface **XMLValidator**

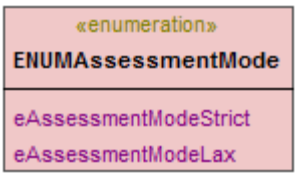
diagram



typedElements	Interface **RaptorXMLFactory** Operation **getXMLValidator**
documentation	Validates the supplied XML document against internal or external DTDs or XML Schemas. Checks well-formedness of XML, DTD, XML Schema, XSLT, and XQuery

	documents.

Enumeration **XMLValidator::ENUMAssessmentMode**

diagram	
typedElements	Interface **XMLValidator** Operation **setAssessmentMode**
documentation	Contains enumeration literals defining the Assessment mode of the XML validator - Strict/Lax.

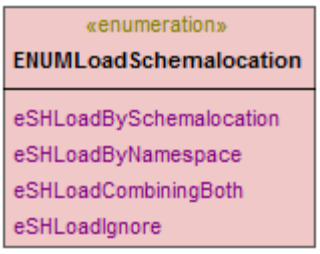
EnumerationLiteral **XMLValidator::ENUMAssessmentMode::eAssessmentModeLax**

documentation	Sets the schema-validity assessment mode to lax.

EnumerationLiteral **XMLValidator::ENUMAssessmentMode::eAssessmentModeStrict**

documentation	Sets the schema-validity assessment mode to strict.

Enumeration **XMLValidator::ENUMLoadSchemalocation**

diagram	
typedElements	Interface **XBRL** Operation **setSchemalocationHints** Interface **XMLValidator** Operation **setSchemalocationHints** Interface **XSLT** Operation **setSchemalocationHints**
documentation	Contains the enumeration literals defining the behaviour of load schema location, which each have an optional namespace attribute and an optional schemaLocation attribute.

EnumerationLiteral **XMLValidator::ENUMLoadSchemalocation::eSHLoadByNamespace**

documentation	Sets the Load Schema Location to LoadByNamespace. Uses the namespace part of xsi:schemaLocation and an empty string in the case of xsi:noNamespaceSchemaLocation and locates the schema via a catalog mapping.

EnumerationLiteral **XMLValidator::ENUMLoadSchemalocation::**

eSHLoadBySchemalocation

documentation	Sets the Load Schema Location to LoadBySchemalocation. Uses the URL of the schema location in the xsi:schemaLocation and xsi:noNamespaceSchemaLocation attributes in XML or XBRL instance documents. This is the default value.

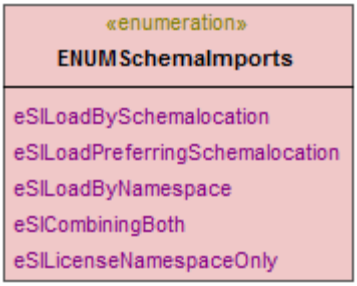
EnumerationLiteral **XMLValidator::ENUMLoadSchemalocation::eSHLoadCombiningBoth**

documentation	Sets the Load Schema Location to CombiningBoth. If either the namespace or URL has a catalog mapping, then the catalog mapping is used. If both have catalog mappings, then the value of the schema-mapping parameter decides which mapping is used. If neither the namespace nor URL has a catalog mapping, the URL is used.

EnumerationLiteral **XMLValidator::ENUMLoadSchemalocation::eSHLoadIgnore**

documentation	Sets the Load Schema Location to LoadIgnore. If the parameter's value is 'ignore', then the xsi:schemaLocation and xsi:noNamespaceSchemaLocation attributes are both ignored.

Enumeration **XMLValidator::ENUMSchemaImports**

diagram	
typedElements	Interface **XBRL** Interface **XMLValidator** Interface **XSLT** Operation **setSchemaImports** Operation **setSchemaImports** Operation **setSchemaImports**
documentation	Contains the enumeration literals defining the behaviour of xs:import elements, which each have an optional namespace attribute and an optional schemaLocation attribute.

EnumerationLiteral **XMLValidator::ENUMSchemaImports::eSICombiningBoth**

documentation	Sets the Schema Import to CombiningBoth. If either the namespace or schemaLocation attribute has a catalog mapping, then the mapping is used, taking account of catalog mappings. If both have catalog mappings, then the value of the --schema-mapping parameter decides which mapping is used. If no catalog mapping is present, the schemaLocation attribute is used.

EnumerationLiteral **XMLValidator::ENUMSchemaImports::eSILicenseNamespaceOnly**

documentation	Sets the Schema Import to LicenseNamespaceOnly. The namespace is imported. No schema document is imported.

EnumerationLiteral **XMLValidator::ENUMSchemaImports::eSILoadByNamespace**

documentation	Sets the Schema Import to LoadByNamespace. The value of the namespace attribute is used to locate the schema via a catalog mapping, taking account of catalog mappings.

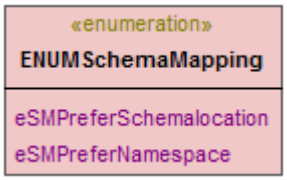
EnumerationLiteral **XMLValidator::ENUMSchemaImports::eSILoadBySchemalocation**

documentation	Sets the Schema Import to LoadBySchemalocation. The value of the schemaLocation attribute is used to locate the schema, taking account of catalog mappings. If the namespace attribute is present, the namespace is imported (licensed).

EnumerationLiteral **XMLValidator::ENUMSchemaImports::eSILoadPreferringSchemalocation**

documentation	Sets the Schema Import to LoadPreferringSchemalocation. If the schemaLocation attribute is present, it is used, taking account of catalog mappings. If no schemaLocation attribute is present, then the value of the namespace attribute is used via a catalog mapping. This is the default value.

Enumeration **XMLValidator::ENUMSchemaMapping**

diagram	
typedElements	Interface **XBRL** Interface **XMLValidator** Interface **XSLT** Operation **setSchemaMapping** Operation **setSchemaMapping** Operation **setSchemaMapping**
documentation	Contains the enumeration literals defining which of the two catalog mappings will be used.

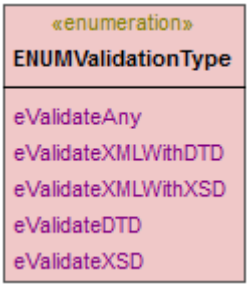
EnumerationLiteral **XMLValidator::ENUMSchemaMapping::eSMPreferNamespace**

documentation	Sets the schema mapping location to PreferNamespace.

EnumerationLiteral **XMLValidator::ENUMSchemaMapping::eSMPreferSchemalocation**

documentation	Sets the schema mapping location to PreferSchemaLocation. The PreferSchemaLocation value refers to the URL mapping

Enumeration **XMLValidator::ENUMValidationType**

diagram	
typedElements	Interface **XMLValidator** Operation **isValid**
documentation	Contains enumeration literals defining how the document will be validated.

EnumerationLiteral **XMLValidator::ENUMValidationType::eValidateAny**

documentation	Sets the validation type to 'any'. This validates any document. The document type is detected automatically.

EnumerationLiteral **XMLValidator::ENUMValidationType::eValidateDTD**

documentation	Sets the validation type to 'validate DTD'. This validates a DTD document.

EnumerationLiteral **XMLValidator::ENUMValidationType::eValidateXMLWithDTD**

documentation	Sets the validation type to 'XML with DTD'. This validates an XML document against a DTD.

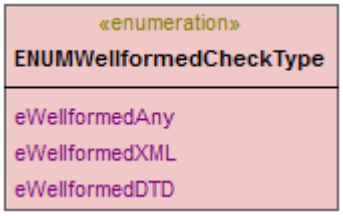
EnumerationLiteral **XMLValidator::ENUMValidationType::eValidateXMLWithXSD**

documentation	Sets the validation type to 'XML with XSD'. This validates an XML document against an XML Schema.

EnumerationLiteral **XMLValidator::ENUMValidationType::eValidateXSD**

documentation	Sets the validation type to 'validate XSD'. This validates a W3C XML Schema document.

Enumeration **XMLValidator::ENUMWellformedCheckType**

diagram	
typedElements	Interface **XMLValidator** Operation **isWellFormed**
documentation	Contains the enumeration literals defining the type of well-formed checks that will be made.

EnumerationLiteral **XMLValidator::ENUMWellformedCheckType::eWellformedAny**

documentation	Sets the well-formed check type to 'Any' This checks an XML or DTD document for well-formedness according to the respective specification/s. The type of document is detected automatically.

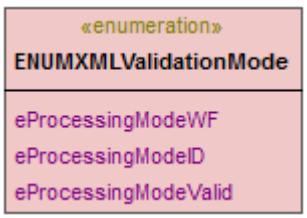
EnumerationLiteral **XMLValidator::ENUMWellformedCheckType::eWellformedDTD**

documentation	Sets the well-formed check type to 'DTD' This checks one or more DTD documents for well-formedness according to the XML 1.0 or XML 1.1 specification. Multiple documents can only be checked by RaptorXML Server, or RaptorXML+XBRL Server.

EnumerationLiteral **XMLValidator::ENUMWellformedCheckType::eWellformedXML**

documentation	Sets the well-formed check type to 'XML' This checks one or more XML documents for well-formedness according to the XML 1.0 or XML 1.1 specification. Multiple documents can only be checked by RaptorXML Server, or RaptorXML+XBRL Server.

Enumeration **XMLValidator::ENUMXMLValidationMode**

diagram	
typedElements	Interface **XMLValidator** Operation **setXMLValidationMode** Interface **XQuery** Operation **setXMLValidationMode** Interface **XSLT** Operation **setXMLValidationMode**

documentation	Contains the enumeration literals defining the XML processing mode that will be used.

EnumerationLiteral **XMLValidator::ENUMXMLValidationMode::eProcessingModeID**

documentation	internal

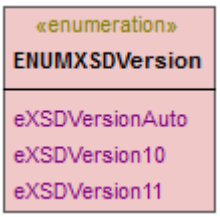
EnumerationLiteral **XMLValidator::ENUMXMLValidationMode::eProcessingModeValid**

documentation	Sets the XML processing mode to 'validation'.

EnumerationLiteral **XMLValidator::ENUMXMLValidationMode::eProcessingModeWF**

documentation	Sets the XML processing mode to 'well-formed'. This is the default value.

Enumeration **XMLValidator::ENUMXSDVersion**

diagram	
typedElements	Interface **XMLValidator** Interface **XQuery** Interface **XSLT** Operation **setXSDVersion** Operation **setXSDVersion** Operation **setXSDVersion**
documentation	Contains enumeration literals defining the XML Schema version that the document will be validated against.

EnumerationLiteral **XMLValidator::ENUMXSDVersion::eXSDVersion10**

documentation	Sets the XML Schema version that the document will be validated against to XML-Schema 1.0.

EnumerationLiteral **XMLValidator::ENUMXSDVersion::eXSDVersion11**

documentation	Sets the XML Schema version that the document will be validated against to XML-Schema 1.1.

EnumerationLiteral **XMLValidator::ENUMXSDVersion::eXSDVersionAuto**

documentation	Sets the XML Schema version that the document will be validated against to 'auto-detect'.

Operation **XMLValidator::getLastErrorMessage**

parameter	name	direction	type	multiplicity	default
	return	**return**	**String**		

documentation	Retrieves the last error message from the XML validator engine. Parameters: none Returns the text of the last error message - of type string.

Operation **XMLValidator::isValid**

parameter	name	direction	type	multiplicity	default
	type	**in**	**ENUMValidationType**		
	return	**return**	**boolean**		
documentation	Result of the XML validation against the XML Schema specified by value of ENUMValidationType. Parameters: 'type' holds the value of the enumeration literal. Raises a RaptorXMLException when an error occurs. If an error occurs during execution, use the getLastErrorMessage operation to access additional information.				

Operation **XMLValidator::isValid**

parameter	name	direction	type	multiplicity	default
	return	**return**	**boolean**		
documentation	Result of the XML validation. Parameters: None Values: true on success, false on failure.				

Operation **XMLValidator::isWellFormed**

parameter	name	direction	type	multiplicity	default
	type	**in**	**ENUMWellformedCheckType**		
	return	**return**	**boolean**		
documentation	Result of the well-formedness check specified by value of ENUMWellformedCheckType. Parameters: None Values: true on success, false on failure. Raises a RaptorXMLException when an error occurs.				

	If an error occurs during execution, use the getLastErrorMessage operation to access additional information.

Operation **XMLValidator::isWellFormed**

parameter	name	direction	type	multiplicity	default
	return	**return**	**boolean**		
documentation	Result of the well-formedness check. Parameters: None Values: true on success, false on failure.				

Operation **XMLValidator::setSchemaImports**

parameter	name	direction	type	multiplicity	default
	opt	**in**	**ENUMSchemaImports**		
	return	**return**	**void**		
documentation	Sets the behaviour of xs:import elements, where each has an optional namespace attribute and an optional schemaLocation attribute. Parameters: 'opt' holds the value of the selected ENUMSchemaImports enumeration literal.				

Operation **XMLValidator::setAssessmentMode**

parameter	name	direction	type	multiplicity	default
	mode	**in**	**ENUMAssessmentMode**		
	return	**return**	**void**		
documentation	Sets the Assessment mode of the XML validator (Strict/Lax/Skip), depends on the ENUMAssessmentMode literals. Parameters: 'mode' holds the value of the selected enumeration literal.				

Operation **XMLValidator::setDTDFileName**

parameter	name	direction	type	multiplicity	default
	filePath	**in**	**String**		
	return	**return**	**void**		
documentation	Sets the external DTD document to use for validation Parameters: 'filePath' is an absolute URL that gives the base location of the DTD to use - of type string. An absolute URL specifies the exact location of the file, e.g. http://www.myWebsite.				

	com/xmlfiles/myInputxml.xml.

Operation **XMLValidator::setDTDFromText**

parameter	name	direction	type	multiplicity	default
	dtdText	**in**	**String**		
	return	**return**	**void**		
documentation	Sets the text value for the external DTD. Parameters: 'dtdText' contains the DTD as text - of type string.				

Operation **XMLValidator::setEnableNamespaces**

parameter	name	direction	type	multiplicity	default
	enable	**in**	**boolean**		
	return	**return**	**void**		
documentation	Enables namespace-aware processing. This is useful for checking the XML instance for errors due to incorrect namespaces. Parameters: 'enable' holds the boolean value - of type boolean. Values: 'true' enables namespace-aware processing, 'false' disables it.				

Operation **XMLValidator::setInputXMLFileCollection**

parameter	name	direction	type	multiplicity	default
	fileCollection	**in**			
	return	**return**	**void**		
documentation	Sets the collection of XML files that will be used as input data. Parameters: 'fileCollection' is a collection of strings containing the absolute URLs of each of the XML files.				

Operation **XMLValidator::setInputXMLFileName**

parameter	name	direction	type	multiplicity	default
	filePath	**in**	**String**		
	return	**return**	**void**		
documentation	Sets the file name (as a URL) of the input XML file. Parameters: 'filePath' is an absolute URL that gives the base location of the XML file - of type string. An absolute URL specifies the exact location of the file, e.g. http://www.myWebsite.com/xmlfiles/myInputxml.xml.				

Operation **XMLValidator::setInputXMLFromText**

parameter	name	direction	type	multiplicity	default
	inputText	**in**	**String**		
	return	**return**	**void**		
documentation	Supplies the contents of the XML input document as text. Parameters: 'inputText' contains the XML data - of type string. Note: Use the absolute URL for the input file name. An absolute URL specifies the exact location of the file, e.g. http://www.myWebsite.com/xmlfiles/myInputxml.xml				

Operation **XMLValidator::setInputXMLTextCollection**

parameter	name	direction	type	multiplicity	default
	stringCollection	**in**			
	return	**return**	**void**		
documentation	Sets the collection of XML files that will be used as input data. Parameters: 'stringCollection' is a collection of strings containing the input document names - no type.				

Operation **XMLValidator::setPythonScriptFile**

parameter	name	direction	type	multiplicity	default
	file	**in**	**String**		
	return	**return**	**void**		
documentation	Sets the file name (as a URL) of the Python script file. Parameters: 'file' is an absolute URL that gives the base location of the file - of type string.				

Operation **XMLValidator::setSchemaFileCollection**

parameter	name	direction	type	multiplicity	default
	fileCollection	**in**			
	return	**return**	**void**		
documentation	Sets the collection of files that will be used as external XML Schemas Parameters: 'fileCollection' is a collection/array of absolute URLs containing the location of the schemas - no type.				

Operation **XMLValidator::setSchemaFileName**

parameter	name	direction	type	multiplicity	default
	filePath	**in**	**String**		
	return	**return**	**void**		
documen	Sets the file name for the external XML Schema.				

tation	Parameters: 'filePath' is the absolute URL of the base location of the Schema - of type string.

Operation **XMLValidator::setSchemaFromText**

parameter	name	direction	type	multiplicity	default
	schemaText	**in**	**String**		
	return	**return**	**void**		
documentation	Supplies the content of the external XML Schema as text. Parameters: 'schemaText' is the string containing the XML Schema as text.				

Operation **XMLValidator::setSchemalocationHints**

parameter	name	direction	type	multiplicity	default
	opt	**in**	**ENUMLoadSchemalocation**		
	return	**return**	**void**		
documentation	Sets the location that the schema will be loaded from, depends on ENUMLoadSchemaLocation. Parameters: 'opt' holds the value of the selected enumeration literal - of type ENUMLoadSchemaLocation.				

Operation **XMLValidator::setSchemaMapping**

parameter	name	direction	type	multiplicity	default
	opt	**in**	**ENUMSchemaMapping**		
	return	**return**	**void**		
documentation	Sets the location that the schema will be loaded from, depends on ENUMSchemaMapping. Parameters: 'opt' holds the value of the selected enumeration literal - of type ENUMSchemaMapping.				

Operation **XMLValidator::setSchemaTextCollection**

parameter	name	direction	type	multiplicity	default
	stringCollection	**in**			
	return	**return**	**void**		
documentation	Sets the collection of strings that will be used as external XML Schemas. Parameters: 'stringCollection' is a collection/array of absolute URLs containing the location of the schemas - no type.				

Operation **XMLValidator::setStreaming**

parameter	name	direction	type	multiplicity	default
	support	**in**	**boolean**		
	return	**return**	**void**		
documentation	Enables streaming validation. In streaming mode, data stored in memory is minimized and processing is faster. Parameters: 'support' holds the boolean value - of type boolean. Values: 'true' enables streaming validation, 'false' disables it. Default is true.				

Operation **XMLValidator::setXincludeSupport**

parameter	name	direction	type	multiplicity	default
	support	**in**	**boolean**		
	return	**return**	**void**		
documentation	Enables or disables the use of XML Inclusions (XInclude elements). Parameters: 'support' holds the boolean value - of type boolean. Values: 'true' enables the XInclude include elements, 'false', ignores them.				

Operation **XMLValidator::setXMLValidationMode**

parameter	name	direction	type	multiplicity	default
	mode	**in**	**ENUMXMLValidationMode**		
	return	**return**	**void**		
documentation	Sets the XML validation mode, depends on ENUMXMLValidationMode. Parameters: 'mode' holds the value of the selected enumeration literal - of type ENUMXMLValidationMode.				

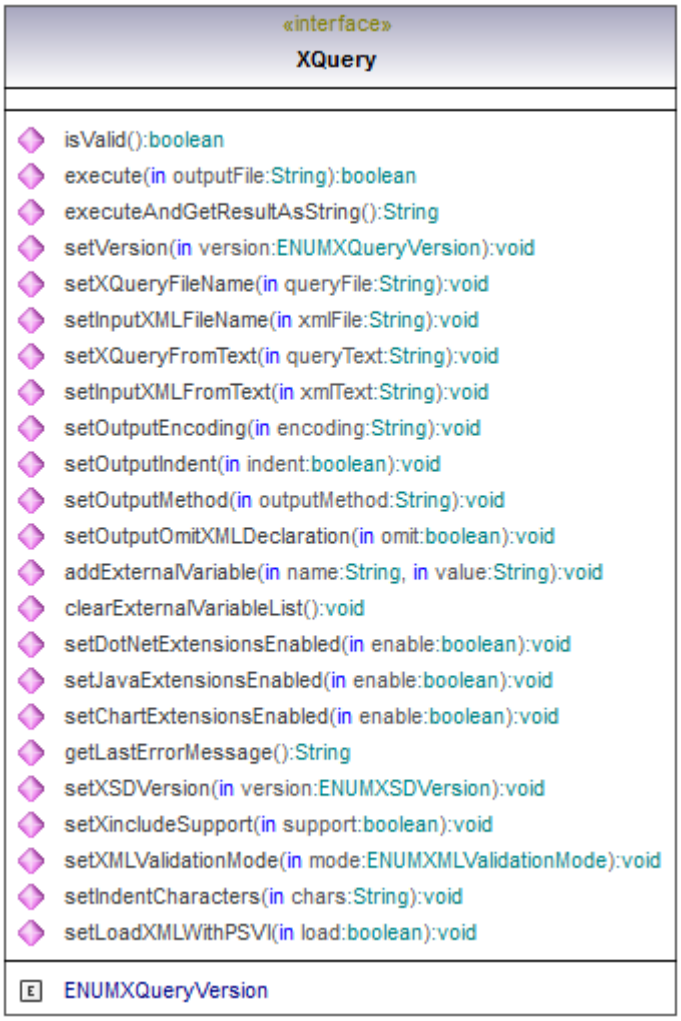
Operation **XMLValidator::setXSDVersion**

parameter	name	direction	type	multiplicity	default
	version	**in**	**ENUMXSDVersion**		
	return	**return**	**void**		
documentation	Defines the XML Schema version that the document will be validated against. Parameters: 'version' holds the XML Schema version set by the EnumXSDVersion literal - of type EnumXSDVersion.				

4.3 RaptorXMLJava - XQuery

Interface **XQuery**

diagram



typedElements

Interface **RaptorXMLFactory** Operation **getXQuery**

documentation

Executes XQuery 1.0 and 3.0 documents using the RaptorXML engine.

XQuery and XML documents can be provided as a file (via a URL) or, in the case of COM usage, as a text string.

Output is returned as a file (at a named location) or, in the case of COM usage, as a text 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.

Where string inputs are to be interpreted as URLs, absolute paths should be used.

Operation **XQuery::addExternalVariable**

parameter	name	direction	type	multiplicity	default
	name	**in**	**String**		
	value	**in**	**String**		
	return	**return**	**void**		

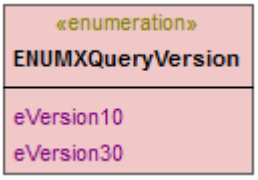
documentation	Adds the name and value of a new external variable. Parameters: 'name' is the variable name, and is a valid QName - of type string. 'value' is the value of the variable - of type string. Each external variable and its value is to be specified in a separate call to the method. Variables 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.

Operation **XQuery::clearExternalVariableList**

parameter	name	direction	type	multiplicity	default
	return	**return**	**void**		

documentation	Clears the external variables list created with the AddExternalVariable method.

Enumeration **XQuery::ENUMXQueryVersion**

diagram	
typedElements	Interface **XQuery** Operation **setVersion**
documentation	Contains enumeration literals defining the XQuery versions you are going to use - XQuery 1.0/3.0.

EnumerationLiteral **XQuery::ENUMXQueryVersion::eVersion10**

documentation	Sets the XQuery version you are going to use to XQuery 1.0.

EnumerationLiteral **XQuery::ENUMXQueryVersion::eVersion30**

documentation	Sets the XQuery version you are going to use to XQuery 3.0.

Operation **XQuery::execute**

parameter	name	direction	type	multiplicity	default
	outputFile	**in**	**String**		
	return	**return**	**boolean**		
documentation	Executes the XQuery transformation (depending on the value of ENUMXQueryVersion) and saves the result to an output file. Parameters: 'outputFile' is the path (and file name) of the result file. If an error occurs during execution, use the getLastErrorMessage operation to access additional information. Raises a RaptorXMLException when an error occurs.				

Operation **XQuery::executeAndGetResultAsString**

parameter	name	direction	type	multiplicity	default
	return	**return**	**String**		
documentation	Executes the XQuery and returns the result as a string. Parameters: none. If an error occurs during execution, use the getLastErrorMessage operation to access additional information. Raises a RaptorXMLException when an error occurs.				

Operation **XQuery::getLastErrorMessage**

parameter	name	direction	type	multiplicity	default
	return	**return**	**String**		
documentation	Retrieves the last error message from the XQuery engine. Parameters: none Returns the text of the last error message - of type string.				

Operation **XQuery::isValid**

parameter	name	direction	type	multiplicity	default
	return	**return**	**boolean**		
documentation	Result of the XQuery 1.0/3.0 validation (depending on the value of ENUMXQueryVersion). Parameters: None Values: true on success, false on failure. Raises a RaptorXMLException when an error occurs.				

Operation **XQuery::setChartExtensionsEnabled**

parameter	name	direction	type	multiplicity	default
	enable	**in**	**boolean**		
	return	**return**	**void**		
documentation	Enables or disables the Altova Chart extensions. Parameters: 'enable' holds the boolean value - of type boolean. Values: 'true' enables the chart extensions, 'false' disables them.				

Operation **XQuery::setDotNetExtensionsEnabled**

parameter	name	direction	type	multiplicity	default
	enable	**in**	**boolean**		
	return	**return**	**void**		
documentation	Enables or disables the Visual Studio .NET extensions. Parameters: 'enable' holds the boolean value - of type boolean. Values: 'true' enables the .NET extensions, 'false' disables them.				

Operation **XQuery::setIndentCharacters**

parameter	name	direction	type	multiplicity	default
	chars	**in**	**String**		
	return	**return**	**void**		
documentation	Sets the character string that will be used as indentation. Parameters: 'chars' holds the indentation character - of type string.				

Operation **XQuery::setInputXMLFileName**

parameter	name	direction	type	multiplicity	default
	xmlFile	**in**	**String**		
	return	**return**	**void**		
documentation	Sets the file name (as a URL) of the input XML instance to be transformed. Parameters: 'xmlFile' holds the name/URL of the XML file - of type string. Note: Use the absolute URL for the input file name. An absolute URL specifies the exact location of the file, e.g. http://www.myWebsite.com/xmlfiles/myInputxml.xml				

Operation **XQuery::setInputXMLFromText**

parameter	name	direction	type	multiplicity	default
	xmlText	**in**	**String**		
	return	**return**	**void**		
documentation	Supplies the contents of the XML input document as text. Parameters: 'xmlText' contains the input XML data - of type string. Note: Use the absolute URL for the input file name. An absolute URL specifies the exact location of the file, e.g. http://www.myWebsite.com/xmlfiles/myInputxml.xml				

Operation **XQuery::setJavaExtensionsEnabled**

parameter	name	direction	type	multiplicity	default
	enable	**in**	**boolean**		
	return	**return**	**void**		
documentation	Enables or disables the Java extensions. Parameters: 'enable' holds the boolean value - of type boolean. Values: 'true' enables the Java extensions, 'false' disables them.				

Operation **XQuery::setLoadXMLWithPSVI**

parameter	name	direction	type	multiplicity	default
	load	**in**	**boolean**		
	return	**return**	**void**		
documentation	Enables or disables the option 'Load post schema validation ínfoset'. Returns the boolean value. Values: 'true' enables the post schema validation infoset, 'false' disables it.				

Operation **XQuery::setOutputEncoding**

parameter	name	direction	type	multiplicity	default
	encoding	**in**	**String**		
	return	**return**	**void**		
documentation	Sets the encoding for the result document. Parameters: 'encoding' is the encoding name (e.g. : UTF-8, UTF-16, ASCII, 8859-1, 1252) - of type string.				

Operation **XQuery::setOutputIndent**

parameter	name	direction	type	multiplicity	default
	indent	**in**	**boolean**		

	return	**return**	**void**
documentation	Enable/disables the indentation option for the result document. Parameters: 'indent' holds the boolean value - of type boolean. Values: 'true' enables indentation, 'false' disables it.		

Operation **XQuery::setOutputMethod**

parameter	name	direction	type	multiplicity	default
	outputMethod	**in**	**String**		
	return	**return**	**void**		
documentation	Sets the serialization method for the result document. Parameters: 'outputMethod' holds the serialization method - of type string. Values: xml, xhtml, html, text.				

Operation **XQuery::setOutputOmitXMLDeclaration**

parameter	name	direction	type	multiplicity	default
	omit	**in**	**boolean**		
	return	**return**	**void**		
documentation	Enables/disables the output of the XML declaration for the result document. Parameters: 'omit' holds the boolean value - of type boolean. Values: 'true' omits the XML declaration, 'false' includes it.				

Operation **XQuery::setVersion**

parameter	name	direction	type	multiplicity	default
	version	**in**	**ENUMXQueryVersion**		
	return	**return**	**void**		
documentation	Sets the XQuery version you are going to use (XQuery 1.0/3.0). Parameters: 'version' holds the version number set by the EnumXQueryVersion enumeration literal eVersion10 or eVersion30.				

Operation **XQuery::setXincludeSupport**

parameter	name	direction	type	multiplicity	default
	support	**in**	**boolean**		
	return	**return**	**void**		

documentation	Enables or disables the use of XML Inclusions (XInclude elements). Returns a boolean value. Values: 'true' enables the XInclude include elements, 'false', ignores them.

Operation **XQuery::setXMLValidationMode**

parameter	name	direction	type	multiplicity	default
	mode	**in**	**ENUMXMLValidationMode**		
	return	**return**	**void**		
documentation	Sets the XML validation mode, depends on ENUMXMLValidationMode. 'mode' holds the value of the selected enumeration literal - of type ENUMXMLValidationMode.				

Operation **XQuery::setXQueryFileName**

parameter	name	direction	type	multiplicity	default
	queryFile	**in**	**String**		
	return	**return**	**void**		
documentation	Sets the file name of the XQuery document. Parameters: 'queryFile' holds an absolute URL giving the base location of the XQuery file - of type string. An absolute URL specifies the exact location of the file, e.g. http://www.myWebsite.com/xmlfiles/myInputxml.xml				

Operation **XQuery::setXQueryFromText**

parameter	name	direction	type	multiplicity	default
	queryText	**in**	**String**		
	return	**return**	**void**		
documentation	Supplies the contents of the XQuery statement as text. Parameters: 'queryText' holds the XQuery statement - of type string.				

Operation **XQuery::setXSDVersion**

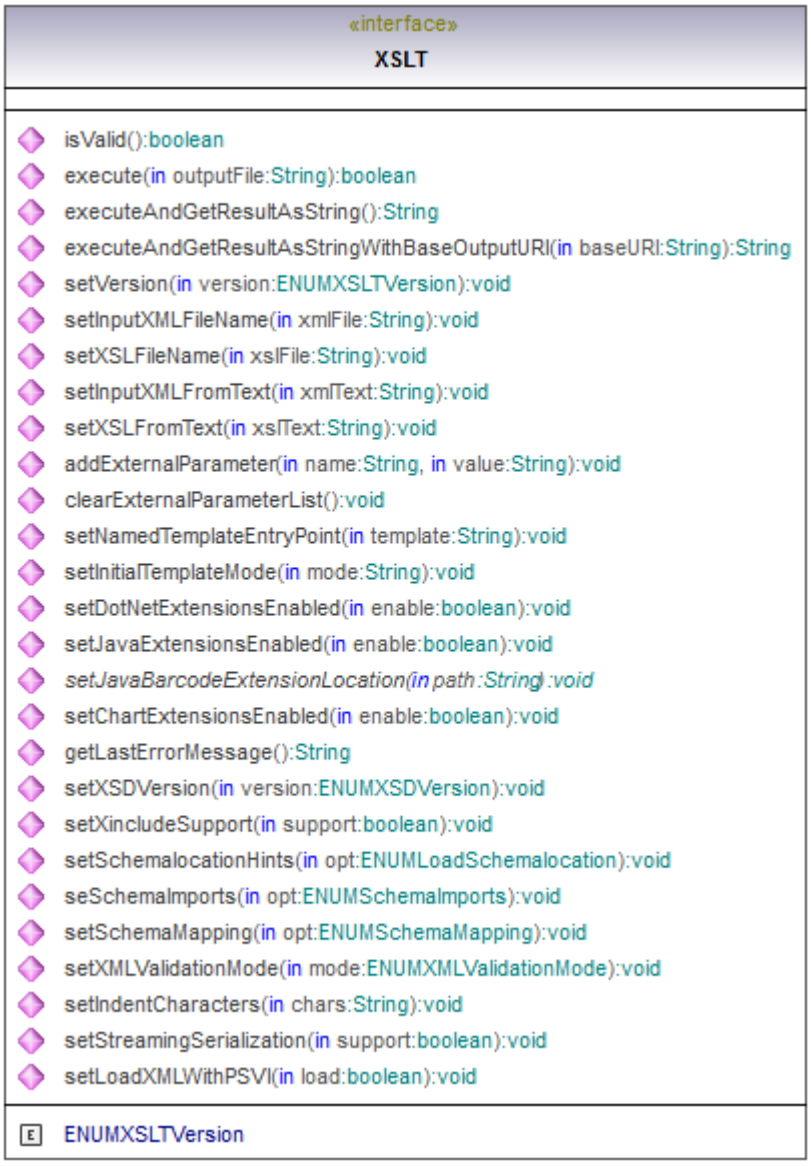
parameter	name	direction	type	multiplicity	default
	version	**in**	**ENUMXSDVersion**		
	return	**return**	**void**		
documentation	Defines the XML Schema version that the document will be validated against. Parameters: 'version' holds the XML Schema version set by the EnumXSDVersion literal - of				

	type EnumXSDVersion.

4.4 RaptorXMLJava - XSLT

Interface **XSLT**

diagram



typedElements	Interface **RaptorXMLFactory** Operation **getXSLT**
documentation	Transforms XML using supplied XSLT 1.0, 2.0, or 3.0 document. XML and XSLT documents can be provided as a file (via a URL) or, in the case of COM usage, as a text string. Output is returned as a file (at a named location) or, in the case of COM usage, as a text string. XSLT parameters can be supplied via the command line and via the COM interface.

	Altova extension functions enable specialized processing, such as for charts.

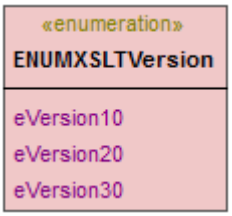
Operation **XSLT::addExternalParameter**

parameter	name	direction	type	multiplicity	default
	name	**in**	**String**		
	value	**in**	**String**		
	return	**return**	**void**		
documentation	Adds the name and value of a new external parameter. Parameters: 'name' is the variable name, and is a valid QName - of type string. 'value' is the value of the variable - of type string. Since parameter values are XPath expressions, parameter values that are strings must be enclosed in single quotes.				

Operation **XSLT::clearExternalParameterList**

parameter	name	direction	type	multiplicity	default
	return	**return**	**void**		
documentation	Clears the external parameters list created with the AddExternalParameters method. Parameters: none				

Enumeration **XSLT::ENUMXSLTVersion**

diagram	
typedElements	Interface **XSLT** Operation **setVersion**
documentation	Contains enumeration literals defining the XSLT versions you are going to use - XSLT 1.0/2.0/3.0.

EnumerationLiteral **XSLT::ENUMXSLTVersion::eVersion10**

documentation	Sets the XSLT version you are going to use to XSLT 1.0.

EnumerationLiteral **XSLT::ENUMXSLTVersion::eVersion20**

documentation	Sets the XSLT version you are going to use to XSLT 2.0.

EnumerationLiteral **XSLT::ENUMXSLTVersion::eVersion30**

documentation	Sets the XSLT version you are going to use to XQuery 3.0.

Operation **XSLT::execute**

parameter	name	direction	type	multiplicity	default
	outputFile	**in**	**String**		
	return	**return**	**boolean**		
documentation	Executes the XSLT transformation (depending on the value of ENUMXSLTVersion) and saves the result to an output file. Parameters: 'outputFile' is the path (and file name) of the result file - of type string. If an error occurs during execution, use the getLastErrorMessage operation to access additional information. Raises a RaptorXMLException when an error occurs.				

Operation **XSLT::executeAndGetResultAsString**

parameter	name	direction	type	multiplicity	default
	return	**return**	**String**		
documentation	Executes the XSLT and returns the result as a string. Parameters: none. If an error occurs during execution, use the getLastErrorMessage operation to access additional information. Raises a RaptorXMLException when an error occurs.				

Operation **XSLT::executeAndGetResultAsStringWithBaseOutputURI**

parameter	name	direction	type	multiplicity	default
	baseURI	**in**	**String**		
	return	**return**	**String**		
documentation	Executes the XSLT and returns the result as a string at the location definied by the base URI. Parameters: none. If an error occurs during execution, use the getLastErrorMessage operation to access additional information. Raises a RaptorXMLException when an error occurs.				

Operation **XSLT::getLastErrorMessage**

parameter	name	direction	type	multiplicity	default
	return	**return**	**String**		
documentation	Retrieves the last error message from the XSLT engine. Parameters: none				

Operation **XSLT::isValid**

parameter	name	direction	type	multiplicity	default
	return	**return**	**boolean**		
documentation	Result of the XSLT validation (depends on the value of ENUMXSLTVersion) Parameters: None Values: true on success, false on failure. Raises a RaptorXMLException when an error occurs.				

Operation **XSLT::setSchemaImports**

parameter	name	direction	type	multiplicity	default
	opt	**in**	**ENUMSchemaImports**		
	return	**return**	**void**		
documentation	Defines the behaviour of xs:import elements, which each have an optional namespace attribute and an optional schemaLocation attribute, depends on ENUMSchemaImports.				

Operation **XSLT::setChartExtensionsEnabled**

parameter	name	direction	type	multiplicity	default
	enable	**in**	**boolean**		
	return	**return**	**void**		
documentation	Enables or disables the Altova Chart extensions. Parameters: 'enable' holds the boolean value - of type boolean. Values: 'true' enables the chart extensions, 'false' disables them.				

Operation **XSLT::setDotNetExtensionsEnabled**

parameter	name	direction	type	multiplicity	default
	enable	**in**	**boolean**		
	return	**return**	**void**		
documentation	Enables or disables the Visual Studio .NET extensions. Parameters:				

	'enable' holds the boolean value - of type boolean. Values: 'true' enables the .NET extensions, 'false' disables them.

Operation **XSLT::setIndentCharacters**

parameter	name	direction	type	multiplicity	default
	chars	**in**	**String**		
	return	**return**	**void**		
documentation	Sets the character string that will be used as indentation. Parameters: 'chars' is the indentation character - of type string.				

Operation **XSLT::setInitialTemplateMode**

parameter	name	direction	type	multiplicity	default
	mode	**in**	**String**		
	return	**return**	**void**		
documentation	Sets the initial mode for XSLT processing. Parameters: 'mode' is the name of the required initial mode - of type string. Templates with this mode value will be processed. For example: SetInitialTemplateMode="MyMode". Note: Transformation must always occur after assigning the XML and XSLT documents.				

Operation **XSLT::setInputXMLFileName**

parameter	name	direction	type	multiplicity	default
	xmlFile	**in**	**String**		
	return	**return**	**void**		
documentation	Sets the file name (as a URL) of the input XML instance to be transformed. Parameters: 'xmlFile' holds the name/URL of the XML file - of type string. Note: Use the absolute URL for the input file name. An absolute URL specifies the exact location of the file, e.g. http://www.myWebsite.com/xmlfiles/myInputxml.xml				

Operation **XSLT::setInputXMLFromText**

parameter	name	direction	type	multiplicity	default
	xmlText	**in**	**String**		
	return	**return**	**void**		
documentation	Supplies the contents of the XML input document as text. Parameters:				

	'xmlText' contains the input XML data - of type string. Note: Use the absolute URL for the input file name. An absolute URL specifies the exact location of the file, e.g. http://www.myWebsite.com/xmlfiles/myInputxml.xml

Operation **XSLT::setJavaBarcodeExtensionLocation**

parameter	name	direction	type	multiplicity	default
	path	**in**	**String**		
	return	**return**	**void**		
documentation	Defines the location of the Java Barcode extension file. Parameters: 'path' holds the location of the extension file - of type string. Use the absolute URL for the input file name. An absolute URL specifies the exact location of the file e.g. e.g. http://www.myWebsite.com/xmlfiles/myInputxml.xml				

Operation **XSLT::setJavaExtensionsEnabled**

parameter	name	direction	type	multiplicity	default
	enable	**in**	**boolean**		
	return	**return**	**void**		
documentation	Enables or disables the Java extensions. Parameters: 'enable' holds the boolean value - of type boolean. Values: 'true' enables the Java extensions, 'false' disables them.				

Operation **XSLT::setLoadXMLWithPSVI**

parameter	name	direction	type	multiplicity	default
	load	**in**	**boolean**		
	return	**return**	**void**		
documentation	Enables or disables the option 'Load post schema validation ínfoset'. Parameters: 'load' holds the boolean value - of type boolean. Values: 'true' enables the post schema validation infoset, 'false' disables them.				

Operation **XSLT::setNamedTemplateEntryPoint**

parameter	name	direction	type	multiplicity	default
	template	**in**	**String**		
	return	**return**	**void**		
documentation	Sets the named template entry point. Parameters:				

	'template' is the name of the node from which processing is to start - of type string.

Operation **XSLT::setSchemalocationHints**

parameter	name	direction	type	multiplicity	default
	opt	**in**	**ENUMLoadSchemalocation**		
	return	**return**	**void**		
documentation	Sets the location that the schema will be loaded from, depends on ENUMLoadSchemaLocation. Parameters: 'opt' holds the value of the selected enumeration literal - of type ENUMLoadSchemaLocation.				

Operation **XSLT::setSchemaMapping**

parameter	name	direction	type	multiplicity	default
	opt	**in**	**ENUMSchemaMapping**		
	return	**return**	**void**		
documentation	Sets the location that the schema will be loaded from, depends on ENUMSchemaMapping. Parameters: 'opt' holds the value of the selected enumeration literal - of type ENUMSchemaMapping.				

Operation **XSLT::setStreamingSerialization**

parameter	name	direction	type	multiplicity	default
	support	**in**	**boolean**		
	return	**return**	**void**		
documentation	Enables streaming serialization. In streaming mode, data stored in memory is minimized and processing is faster. Parameters: 'support' holds the boolean value - of type boolean. Values: 'true' enables streaming serialization, 'false' disables it. Default is true.				

Operation **XSLT::setVersion**

parameter	name	direction	type	multiplicity	default
	version	**in**	**ENUMXSLTVersion**		
	return	**return**	**void**		
documentation	Sets the XQuery version you are going to use (XQuery 1.0/2.0/3.0).				

	Parameters: 'version' holds the version number set by the EnumXSLTVersion enumeration literals - of type EnumXSLTVersion.

Operation **XSLT::setXincludeSupport**

parameter	name	direction	type	multiplicity	default
	support	**in**	**boolean**		
	return	**return**	**void**		
documentation	Enables or disables the use of XML Inclusions (XInclude elements). Parameters: 'support' holds the boolean value - of type boolean. Values: 'true' enables the XInclude include elements, 'false', ignores them.				

Operation **XSLT::setXMLValidationMode**

parameter	name	direction	type	multiplicity	default
	mode	**in**	**ENUMXMLValidationMode**		
	return	**return**	**void**		
documentation	Sets the XML validation mode, depends on ENUMXMLValidationMode. Parameters: 'mode' holds the value of the selected enumeration literal - of type ENUMXMLValidationMode.				

Operation **XSLT::setXSDVersion**

parameter	name	direction	type	multiplicity	default
	version	**in**	**ENUMXSDVersion**		
	return	**return**	**void**		
documentation	Defines the XML Schema version that the document will be validated against. Parameters: 'version' holds the XML Schema version set by the EnumXSDVersion literal - of type EnumXSDVersion.				

Operation **XSLT::setXSLFileName**

parameter	name	direction	type	multiplicity	default
	xslFile	**in**	**String**		
	return	**return**	**void**		
documentation	Sets the path/URL to locate the XSLT file to be used for the transformation. Parameters: 'xslFile' is the path/file name of the XSLT file - of type string. Note:				

	Use the absolute URL for the input file name. An absolute URL specifies the exact location of the file, e.g. http://www.myWebsite.com/xmlfiles/myInputxml.xml.

Operation **XSLT::setXSLFromText**

parameter	name	direction	type	multiplicity	default
	xslText	**in**	**String**		
	return	**return**	**void**		
documentation	Sets the XSLT file name to be used for the transformation. Parameters: 'xslText' is the XML text file name of the XSLT file - of type string.				

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 06/07/13 15:46:38

4.5 RaptorXMLJava - RaptorXMLException

Operation **RaptorXMLException::RaptorXMLException**

parameter	name	direction	type	multiplicity	default
	message	**in**	**String**		

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 06/07/13 15:46:38

Chapter 5

COM / .NET Interface

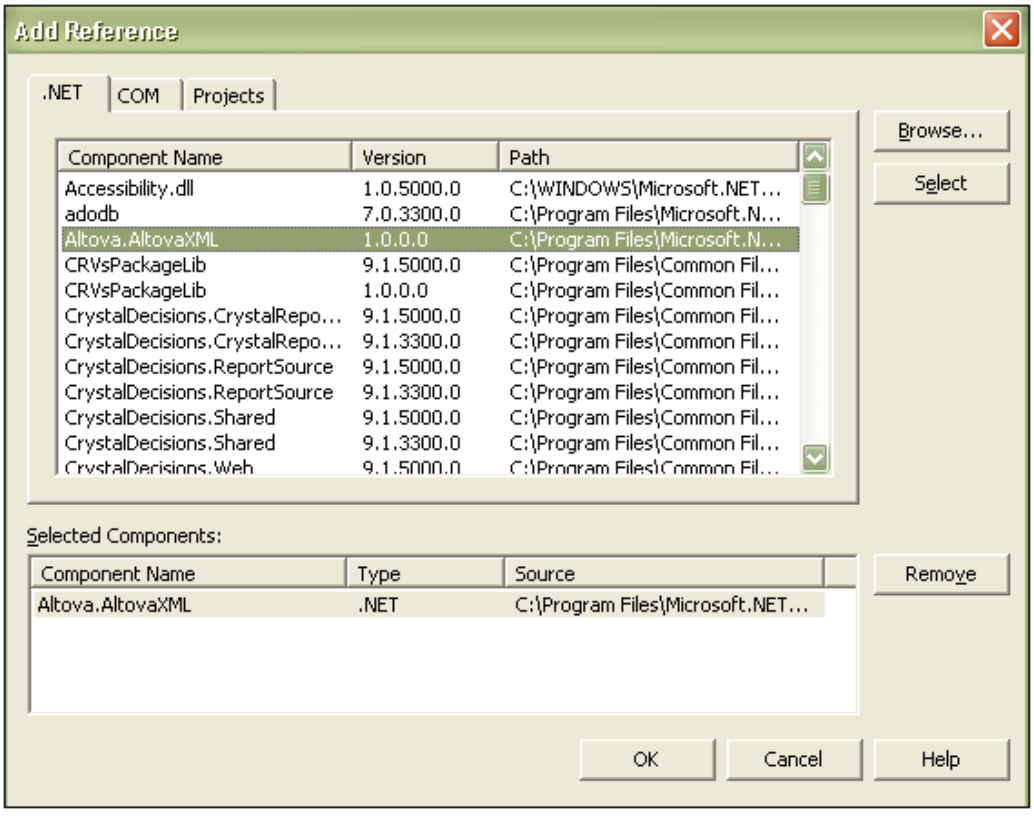
5 COM / .NET Interface

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

Adding the RaptorXML DLL as a reference to the project

The RaptorXML package contains a signed DLL file, named `Altova.RaptorXML.dll`, which will automatically be added to the global assembly cache (and the .NET reference library) when RaptorXML is installed using the RaptorXML 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 RaptorXML component is not in the .NET tab list, it can be selected from the COM tab.)



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

Registering RaptorXML as a COM server object

COM registration is done automatically by the RaptorXML Installer. If you change the location of the file `RaptorXML_COM.exe` after installation, you should register RaptorXML as a COM server object by running the command `RaptorXML_COM.exe /regserver`. (Note that the correct path

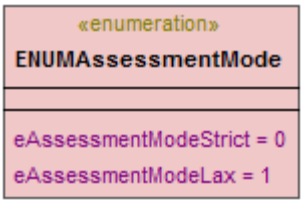
to the `RaptorXML_COM.exe` must be entered. See Registering RaptorXML as a COM Server Object for more details.)

Once the `Altova.RaptorXML.dll` is available to the .NET interface and RaptorXML has been registered as a COM server object, RaptorXML 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 RaptorXML.

5.1 RaptorXMLDev_COM - ENUMAssessmentMode

Enumeration **ENUMAssessmentMode**

diagram	
typedElements	Interface **IXMLValidator** Operation **AssessmentMode**
documentation	Contains enumeration literals defining the Assessment mode of the XML validator - Strict/Lax.

EnumerationLiteral **ENUMAssessmentMode::eAssessmentModeLax**

documentation	Sets the schema-validity assessment mode to lax.

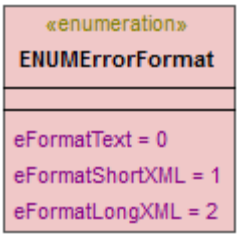
EnumerationLiteral **ENUMAssessmentMode::eAssessmentModeStrict**

documentation	Sets the schema-validity assessment mode to strict.

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 06/07/13 15:48:10

5.2 RaptorXMLDev_COM - ENUMErrorFormat

Enumeration **ENUMErrorFormat**

diagram	
typedElements	Interface **IApplication** Operation **ErrorFormat**
documentation	Contains the enumeration literals defining the format of the error output.

EnumerationLiteral **ENUMErrorFormat::eFormatLongXML**

documentation	Sets the error output format to Long XML. This XML output format provides the most detail of the output formats.

EnumerationLiteral **ENUMErrorFormat::eFormatShortXML**

documentation	Sets the error output format to Short XML. This XML output format is an abbreviated form of the XML Long format.

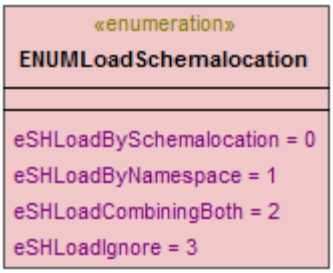
EnumerationLiteral **ENUMErrorFormat::eFormatText**

documentation	Sets the error output format to "Text". This is the default setting.

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 06/07/13 15:48:10

5.3 RaptorXMLDev_COM - ENUMLoadSchemalocation

Enumeration **ENUMLoadSchemalocation**

diagram	
typedElements	Interface **IXBRL** Interface **IXMLValidator** Interface **IXSLT** Operation **SchemalocationHints** Operation **SchemalocationHints** Operation **SchemalocationHints**
documentation	Contains the enumeration literals defining the behaviour of load schema location, which each have an optional namespace attribute and an optional schemaLocation attribute.

EnumerationLiteral **ENUMLoadSchemalocation::eSHLoadByNamespace**

documentation	Sets the Load Schema Location to LoadByNamespace. Uses the namespace part of xsi:schemaLocation and an empty string in the case of xsi:noNamespaceSchemaLocation and locates the schema via a catalog mapping.

EnumerationLiteral **ENUMLoadSchemalocation::eSHLoadBySchemalocation**

documentation	Sets the Load Schema Location to LoadBySchemalocation. Uses the URL of the schema location in the xsi:schemaLocation and xsi:noNamespaceSchemaLocation attributes in XML or XBRL instance documents. This is the default value.

EnumerationLiteral **ENUMLoadSchemalocation::eSHLoadCombiningBoth**

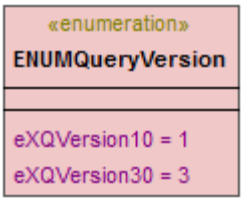
documentation	Sets the Load Schema Location to CombiningBoth. If either the namespace or URL has a catalog mapping, then the catalog mapping is used. If both have catalog mappings, then the value of the schema-mapping parameter decides which mapping is used. If neither the namespace nor URL has a catalog mapping, the URL is used.

EnumerationLiteral **ENUMLoadSchemalocation::eSHLoadIgnore**

documentation	Sets the Load Schema Location to LoadIgnore. If the parameter's value is 'ignore', then the xsi:schemaLocation and xsi:noNamespaceSchemaLocation attributes are both ignored.

5.4 RaptorXMLDev_COM - ENUMQueryVersion

Enumeration **ENUMQueryVersion**

diagram	
documentation	Contains enumeration literals defining the XQuery versions you are going to use - XQuery 1.0/3.0.

EnumerationLiteral **ENUMQueryVersion::eXQVersion10**

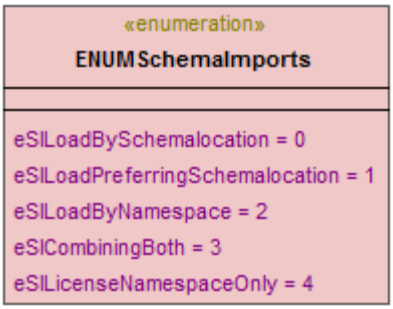
documentation	Sets the XQuery version you are going to use to XQuery 1.0.

EnumerationLiteral **ENUMQueryVersion::eXQVersion30**

documentation	Sets the XQuery version you are going to use to XQuery 3.0.

5.5 RaptorXMLDev_COM - ENUMSchemaImports

Enumeration **ENUMSchemaImports**

diagram	
typedElements	Interface **IXBRL** Interface **IXMLValidator** Interface **IXSLT** Operation **SchemaImports** Operation **SchemaImports** Operation **SchemaImports**
documentation	Contains the enumeration literals defining the behaviour of xs:import elements, which each have an optional namespace attribute and an optional schemaLocation attribute.

EnumerationLiteral **ENUMSchemaImports::eSICombiningBoth**

documentation	Sets the Schema Import to CombiningBoth. If either the namespace or schemaLocation attribute has a catalog mapping, then the mapping is used, taking account of catalog mappings. If both have catalog mappings, then the value of the --schema-mapping parameter decides which mapping is used. If no catalog mapping is present, the schemaLocation attribute is used.

EnumerationLiteral **ENUMSchemaImports::eSILicenseNamespaceOnly**

documentation	Sets the Schema Import to LicenseNamespaceOnly. The namespace is imported. No schema document is imported.

EnumerationLiteral **ENUMSchemaImports::eSILoadByNamespace**

documentation	Sets the Schema Import to LoadByNamespace. The value of the namespace attribute is used to locate the schema via a catalog mapping, taking account of catalog mappings.

EnumerationLiteral **ENUMSchemaImports::eSILoadBySchemalocation**

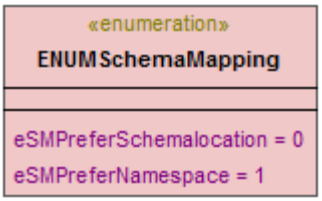
documentation	Sets the Schema Import to LoadBySchemalocation. The value of the schemaLocation attribute is used to locate the schema, taking account of catalog mappings. If the namespace attribute is present, the namespace is imported (licensed).

EnumerationLiteral **ENUMSchemaImports::eSILoadPreferringSchemalocation**

documentation	Sets the Schema Import to LoadPreferringSchemalocation. If the schemaLocation attribute is present, it is used, taking account of catalog mappings. If no schemaLocation attribute is present, then the value of the namespace attribute is used via a catalog mapping. This is the default value.

5.6 RaptorXMLDev_COM - ENUMSchemaMapping

Enumeration **ENUMSchemaMapping**

diagram	
typedElements	Interface **IXBRL** Interface **IXMLValidator** Interface **IXSLT** Operation **SchemaMapping** Operation **SchemaMapping** Operation **SchemaMapping**
documentation	Contains the enumeration literals defining which of the two catalog mappings will be used.

EnumerationLiteral **ENUMSchemaMapping::eSMPreferNamespace**

documentation	Sets the schema mapping location to PreferNamespace.

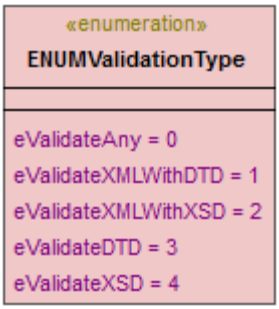
EnumerationLiteral **ENUMSchemaMapping::eSMPreferSchemalocation**

documentation	Sets the schema mapping location to PreferSchemaLocation. The PreferSchemaLocation value refers to the URL mapping

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 06/07/13 15:48:10

5.7 RaptorXMLDev_COM - ENUMValidationType

Enumeration **ENUMValidationType**

diagram	
typedElements	Interface **IXMLValidator** Operation **IsValid**
documentation	Contains enumeration literals defining the type of document that will be validated.

EnumerationLiteral **ENUMValidationType::eValidateAny**

documentation	Sets the validation type to 'any'. This validates any document. The document type is detected automatically.

EnumerationLiteral **ENUMValidationType::eValidateDTD**

documentation	Sets the validation type to 'validate DTD'. This validates a DTD document.

EnumerationLiteral **ENUMValidationType::eValidateXMLWithDTD**

documentation	Sets the validation type to 'XML with DTD'. This validates an XML document against a DTD.

EnumerationLiteral **ENUMValidationType::eValidateXMLWithXSD**

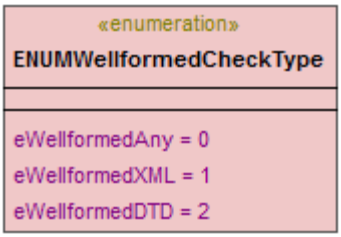
documentation	Sets the validation type to 'XML with XSD'. This validates an XML document against an XML Schema.

EnumerationLiteral **ENUMValidationType::eValidateXSD**

documentation	Sets the validation type to 'validate XSD'. This validates a W3C XML Schema document.

5.8 RaptorXMLDev_COM - ENUMWellformedCheckType

Enumeration **ENUMWellformedCheckType**

diagram	
typedElements	Interface **IXMLValidator** Operation **IsWellFormed**
documentation	Contains the enumeration literals defining the type of well-formed checks that will be made.

EnumerationLiteral **ENUMWellformedCheckType::eWellformedAny**

documentation	Sets the well-formed check type to 'Any' This checks an XML or DTD document for well-formedness according to the respective specification/s. The type of document is detected automatically.

EnumerationLiteral **ENUMWellformedCheckType::eWellformedDTD**

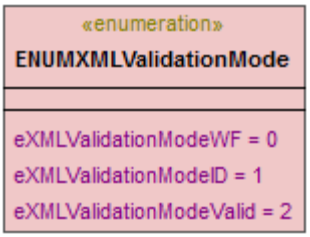
documentation	Sets the well-formed check type to 'DTD' This checks one or more DTD documents for well-formedness according to the XML 1.0 or XML 1.1 specification. Multiple documents can only be checked by RaptorXML Server, or RaptorXML+XBRL Server.

EnumerationLiteral **ENUMWellformedCheckType::eWellformedXML**

documentation	Sets the well-formed check type to 'XML' This checks one or more XML documents for well-formedness according to the XML 1.0 or XML 1.1 specification. Multiple documents can only be checked by RaptorXML Server, or RaptorXML+XBRL Server.

5.9 RaptorXMLDev_COM - ENUMXMLValidationMode

Enumeration **ENUMXMLValidationMode**

diagram	
typedElements	Interface **IXMLValidator** Interface **IXQuery** Interface **IXSLT** Operation **XMLValidationMode** Operation **XMLValidationMode** Operation **XMLValidationMode**
documentation	Contains the enumeration literals defining the XML processing mode that will be used.

EnumerationLiteral **ENUMXMLValidationMode::eXMLValidationModeID**

documentation	internal

EnumerationLiteral **ENUMXMLValidationMode::eXMLValidationModeValid**

documentation	Sets the XML processing mode to 'validation'.

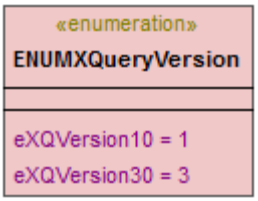
EnumerationLiteral **ENUMXMLValidationMode::eXMLValidationModeWF**

documentation	Sets the XML processing mode to 'well-formed'. This is the default value.

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 06/07/13 15:48:10

5.10 RaptorXMLDev_COM - ENUMXQueryVersion

Enumeration **ENUMXQueryVersion**

diagram	
typedElements	Interface **IXQuery** Operation **EngineVersion**
documentation	Contains enumeration literals defining the XQuery versions you are going to use - XQuery 1.0/3.0.

EnumerationLiteral **ENUMXQueryVersion::eXQVersion10**

documentation	Sets the XQuery version you are going to use to XQuery 1.0.

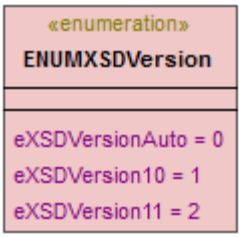
EnumerationLiteral **ENUMXQueryVersion::eXQVersion30**

documentation	Sets the XQuery version you are going to use to XQuery 3.0.

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 06/07/13 15:48:10

5.11 RaptorXMLDev_COM - ENUMXSDVersion

Enumeration **ENUMXSDVersion**

diagram	
typedElements	Interface **IXMLValidator** Interface **IXQuery** Interface **IXSLT** Operation **XSDVersion** Operation **XSDVersion** Operation **XSDVersion**
documentation	Contains enumeration literals defining the XML Schema version that the document will be validated against.

EnumerationLiteral **ENUMXSDVersion::eXSDVersion10**

documentation	Sets the XML Schema version that the document will be validated against to XML-Schema 1.0.

EnumerationLiteral **ENUMXSDVersion::eXSDVersion11**

documentation	Sets the XML Schema version that the document will be validated against to XML-Schema 1.1.

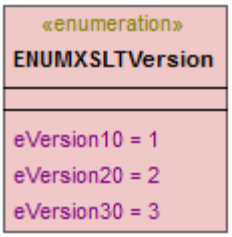
EnumerationLiteral **ENUMXSDVersion::eXSDVersionAuto**

documentation	Sets the XML Schema version that the document will be validated against to 'auto-detect'.

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 06/07/13 15:48:10

5.12 RaptorXMLDev_COM - ENUMXSLTVersion

Enumeration **ENUMXSLTVersion**

diagram	
typedElements	Interface **IXSLT** Operation **EngineVersion**
documentation	Contains enumeration literals defining the XSLT versions you are going to use - XSLT 1.0/2.0/3.0.

EnumerationLiteral **ENUMXSLTVersion::eVersion10**

documentation	Sets the XSLT version you are going to use to XSLT 1.0.

EnumerationLiteral **ENUMXSLTVersion::eVersion20**

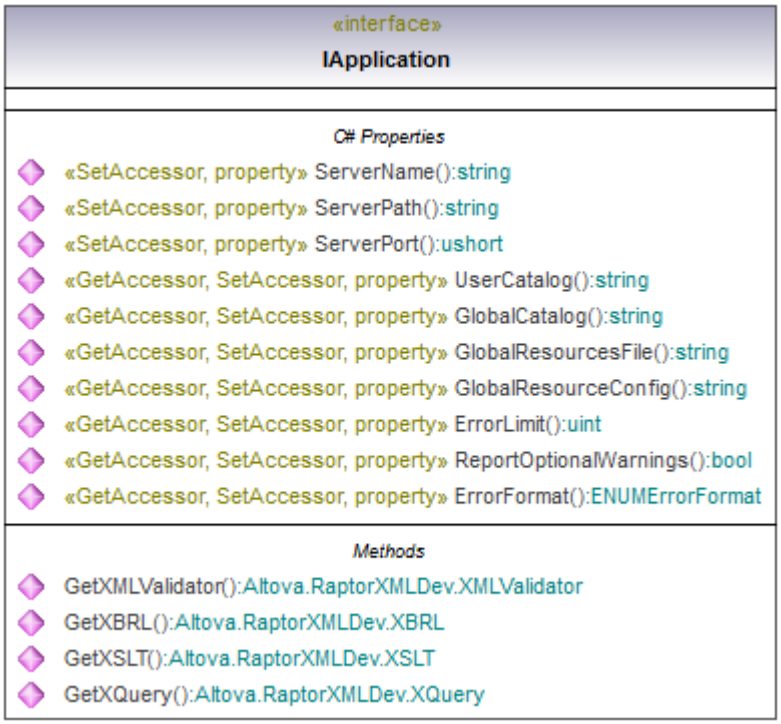
documentation	Sets the XSLT version you are going to use to XSLT 2.0.

EnumerationLiteral **ENUMXSLTVersion::eVersion30**

documentation	Sets the XSLT version you are going to use to XSLT 3.0.

5.13 RaptorXMLDev_COM - IApplication

Interface **IApplication**

diagram	
documentation	Description of IApplication interface

Operation **IApplication::ErrorFormat**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMErrorFormat**			
documentation	Sets the RaptorXML error format (Text, ShortXML, LongXML), depends on the ENUMErrorFormat literals. Parameters: Returns the value of the selected enumeration literal, - of type ENUMErrorFormat.					

Operation **IApplication::ErrorLimit**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**uint**			
documentation	Configures the RaptorXML validation error limit property. Properties: Defines the number of errors to be reported before halting execution - of type uint.					

Operation **IApplication::GetXBRL**

parameter	name	direction	type	type modifier	multiplicity	default

r	**return**	**return**	**Altova.RaptorXMLDev.XBRL**
documentation	Retrieves the XBRL engine and returns a new XBRL instance of RaptorXMLFactory. Only supported by RaptorXML+XBRL Server. Properties: Returns an instance of the XBRL engine.		

Operation **IApplication::GetXMLValidator**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**Altova.RaptorXMLDev.XMLValidator**			
documentation	Retrieves the XML engine and returns a new XML validator instance of RaptorXMLFactory. Properties: Returns an instance of the XML validator engine.					

Operation **IApplication::GetXQuery**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**Altova.RaptorXMLDev.XQuery**			
documentation	Retrieves the XQuery engine and returns a new XQuery instance of RaptorXMLFactory. Properties: Returns an instance of the XQuery engine.					

Operation **IApplication::GetXSLT**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**Altova.RaptorXMLDev.XSLT**			
documentation	Retrieves the XSLT engine and returns a new XSLT instance of XMLFactory. Properties: Returns an instance of the XSLT engine.					

Operation **IApplication::GlobalCatalog**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			

documentation	Sets the file name (as a URL) of the global catalog (e.g. RootCatalog.xml). Properties: The URL for the base location of the global catalog file - of type string. Note: Use the absolute URL for the input file name. An absolute URL specifies the exact location of the file, e.g. http://www.myWebsite.com/xmlfiles/myGlobCatalog.xml

Operation **IApplication::GlobalResourceConfig**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			
documentation	Sets the active configuration of the global resource. Properties: The name of the configuration used by the active global resource - of type string.					

Operation **IApplication::GlobalResourcesFile**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			
documentation	Sets the file name/URL for the global resource XML file (e.g. GlobalResources.xml). Properties: The name of the XML file - of type string. Note: Use the absolute URL for the input file name. An absolute URL specifies the exact location of the file, e.g. http://www.myWebsite.com/xmlfiles/myInputxml.xml					

Operation **IApplication::ReportOptionalWarnings**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			
documentation	Enables / disables the reporting of warnings. Properties: Returns a boolean value. 'true' enables reporting, 'false' disables it.					

Operation **IApplication::ServerName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			
documentation	Sets the server name for the HTTP server (this method fails in RaptorXML Development Edition). Properties: The server name - of type string. Raises a RaptorXMLException when an error occurs.					

Operation **IApplication::ServerPath**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			
documentation	Sets the server path as a URL for the HTTP server (this method fails in RaptorXML Development Edition). Properties: The server path - of type string. Raises a RaptorXMLException when an error occurs.					

Operation **IApplication::ServerPort**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ushort**			
documentation	Sets the server port for the HTTP server (this method fails in RaptorXML Development Edition). Properties: The server port - of type ushort. Raises a RaptorXMLException when an error occurs.					

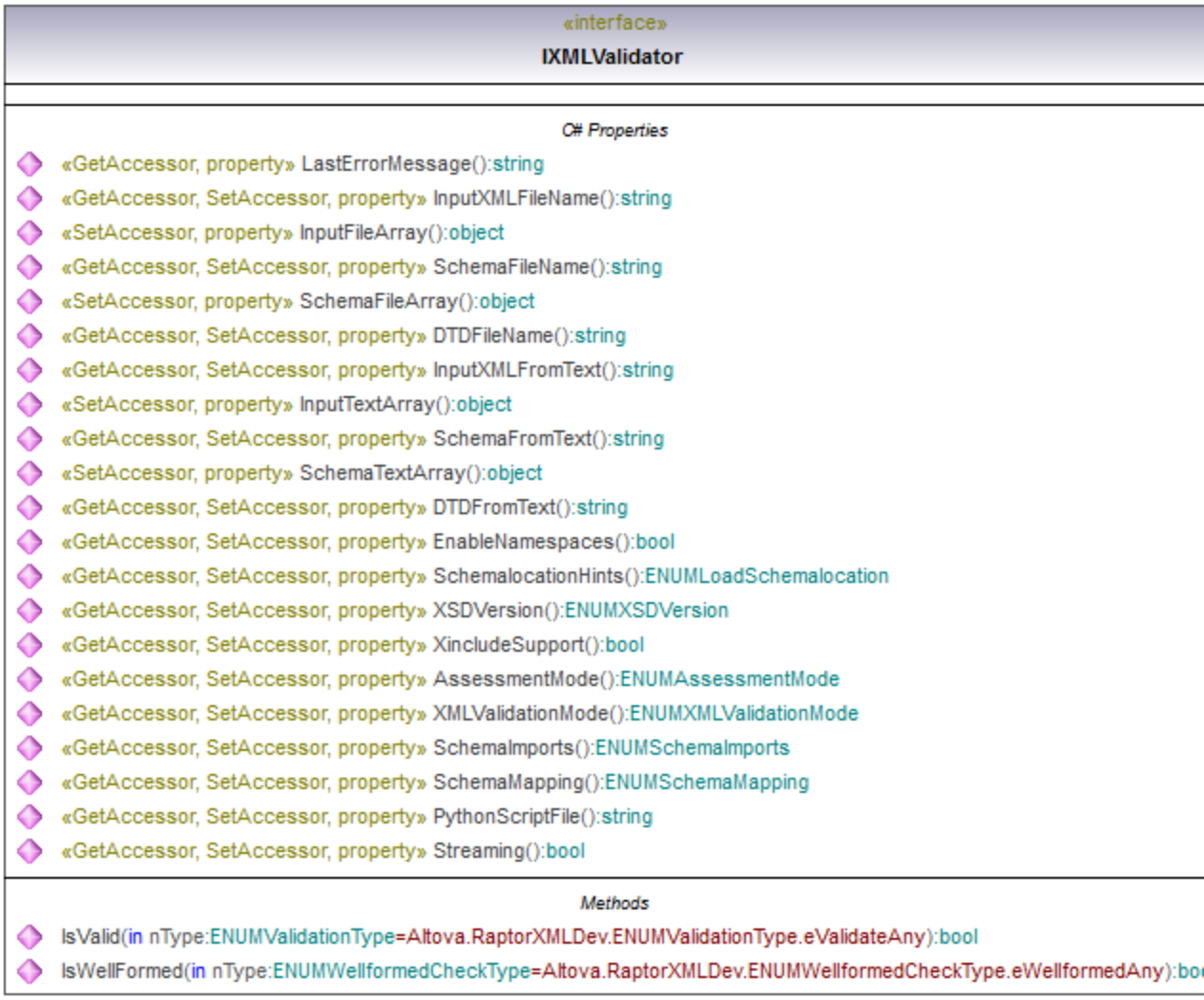
Operation **IApplication::UserCatalog**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			
documentation	Sets the name (as a URL) for the user-defined catalog (e.g. CustomCatalog.xml). Properties: The name of the user-defined catalog - of type string. Note: Use the absolute URL for the input file name. An absolute URL specifies the exact location of the file.					

5.14 RaptorXMLDev_COM - IXMLValidator

Interface **IXMLValidator**

diagram



documentation

Description
The IXMLValidator 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.

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.

Operation **IXMLValidator::AssessmentMode**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMAsses**			

	smentMode
documentation	Sets the Assessment mode of the XML validator (Strict/Lax/Skip), depends on the ENUMAssessmentMode literals. Properties: The value of the selected enumeration literal.

Operation **IXMLValidator::DTDFileName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			
documentation	Sets the external DTD document to use for validation Properties: An absolute URL that gives the base location of the DTD to use - of type string. An absolute URL specifies the exact location of the file, e.g. http://www.myWebsite.com/xmlfiles/myInputxml.xml.					

Operation **IXMLValidator::DTDFromText**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			
documentation	Sets the text value for the external DTD. Properties: Contains the DTD as text - of type string.					

Operation **IXMLValidator::EnableNamespaces**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			
documentation	Enables namespace-aware processing. This is useful for checking the XML instance for errors due to incorrect namespaces. Properties: Returns a boolean value. 'true' enables namespace-aware processing, 'false' disables it.					

Operation **IXMLValidator::InputFileArray**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**object**			
documentation	Sets the array of XML files that will be used as input data. Properties: Object containing the strings of the absolute URLs of each of the XML files.					

Operation **IXMLValidator::InputTextArray**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**object**			

documentation	Sets the array of text files that will be used as input data. Properties: Object containing the strings of the absolute URLs of each of the text input files.

Operation **IXMLValidator::InputXMLFileName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			
documentation	Sets the file name (as a URL) of the input XML file. Properties: An absolute URL that gives the base location of the XML file - of type string. An absolute URL specifies the exact location of the file, e.g. http://www.myWebsite.com/xmlfiles/myInputxml.xml.					

Operation **IXMLValidator::InputXMLFromText**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			
documentation	Supplies the contents of the XML input document as text. Properties: Contains the XML data - of type string. Note: Use the absolute URL for the input file name. An absolute URL specifies the exact location of the file, e.g. http://www.myWebsite.com/xmlfiles/myInputxml.xml					

Operation **IXMLValidator::IsValid**

parameter	name	direction	type	type modifier	multiplicity	default
	nType	**in**	**ENUMValidationType**			**Altova.RaptorXMLDev.ENUMValidationType.eValidateAny**
	return	**return**	**bool**			
documentation	Result of the XML validation. Properties: 'nType' the value of the ENUMValidationType. Returns: 'true' on success, 'false' on failure. Raises a RaptorXMLException when an error occurs. If an error occurs during execution, use the LastErrorMessage operation to access additional information.					

Operation **IXMLValidator::IsWellFormed**

parameter	name	direction	type	type modifier	multiplicity	default
	nType	**in**	**ENUMWellformedCheckType**			**Altova.RaptorXMLDev.ENUMWellformedCheckType.eWellformedAny**
	return	**return**	**bool**			
documentation	Result of the well-formedness check specified by the value of ENUMWellformedCheckType. Properties: 'nType' is the value of ENUMWellformedCheckType. Returns: 'true' on success, 'false' on failure. Raises a RaptorXMLException when an error occurs. If an error occurs during execution, use the LastErrorMessage operation to access additional information.					

Operation **IXMLValidator::LastErrorMessage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			
documentation	Retrieves the last error message from the XML engine. Properties: Returns the text of the last error message - of type string.					

Operation **IXMLValidator::PythonScriptFile**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			
documentation	Sets the file name (as a URL) of the Python script file. Returns an absolute URL that gives the base location of the file - of type string.					

Operation **IXMLValidator::SchemaFileArray**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**object**			
documentation	Sets the array of files that will be used as external XML Schemas. Properties: The array of absolute URLs containing the location of the schemas - of type object.					

Operation **IXMLValidator::SchemaFileName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			
documentation	Sets the file name for the external XML Schema. Properties: The absolute URL of the base location of the Schema - of type string.					

Operation **IXMLValidator::SchemaFromText**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			
documentation	Supplies the content of the external XML Schema as text. Properties: The string containing the XML Schema as text - of type string.					

Operation **IXMLValidator::SchemaImports**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMSchemaImports**			
documentation	Specifies the behaviour of xs:import elements, which each has an optional namespace attribute and an optional schemaLocation attribute. Returns the schema import method of type EnumSchemaImports					

Operation **IXMLValidator::SchemaLocationHints**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMLoadSchemalocation**			
documentation	Sets the location that the schema will be loaded from, depends on ENUMLoadSchemaLocation. Returns the value of the selected enumeration literal - of type ENUMLoadSchemaLocation.					

Operation **IXMLValidator::SchemaMapping**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMSchemaMapping**			
documentation	Sets the location that the schema will be loaded from, depends on ENUMSchemaMapping. Returns the value of the selected enumeration literal - of type ENUMSchemaMapping.					

Operation **IXMLValidator::SchemaTextArray**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**object**			
documentation	Sets the collection of strings that will be used as external XML Schemas. Properties: An array of absolute URLs containing the location of the schemas - type object.					

Operation **IXMLValidator::Streaming**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			
documentation	Enables streaming validation. In streaming mode, data stored in memory is minimized and processing is faster. Values: 'true' enables streaming validation, 'false' disables it. Default is true.					

Operation **IXMLValidator::XincludeSupport**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			
documentation	Enables or disables the use of XML Inclusions (XInclude elements). Properties: Returns a boolean value. 'true' enables the XInclude include elements, 'false', ignores them.					

Operation **IXMLValidator::XMLValidationMode**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMXMLValidationMode**			
documentation	Sets the XML validation mode, depends on ENUMXMLValidationMode. Returns the value of the selected enumeration literal - of type ENUMXMLValidationMode.					

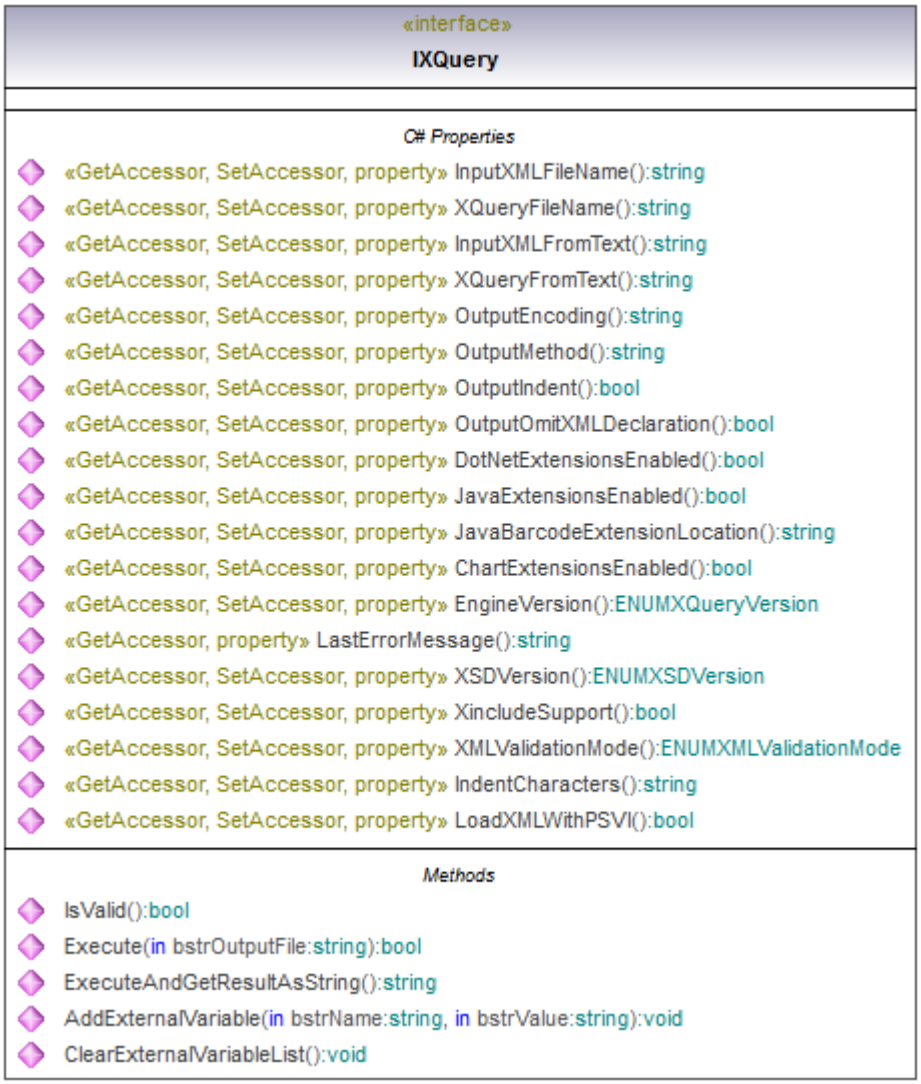
Operation **IXMLValidator::XSDVersion**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMXSDVersion**			
documentation	Defines the XML Schema version that the document will be validated against. Properties: The XML Schema version set by the EnumXSDVersion literal - of type EnumXSDVersion.					

5.15 RaptorXMLDev_COM - IXQuery

Interface **IXQuery**

diagram



documentation

The IXQuery interface provides methods and properties to execute an XQuery 1.0/3.0 transformation using the RaptorXML 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.

Operation **IXQuery::AddExternalVariable**

parameter	name	direction	type	type modifier	multiplicity	default
	bstrName	**in**	**string**			
	bstrValue	**in**	**string**			
	return	**return**	**void**			
documentation	Adds the name and value of a new external variable. Properties: 'bstrName' is the variable name, and is a valid QName - of type string. 'bstrValue' is the value of the variable - of type string. 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.					

Operation **IXQuery::ChartExtensionsEnabled**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			
documentation	Enables or disables the Altova Chart extensions. Properties: Returns a boolean value. 'true' enables the chart extensions, 'false' disables them.					

Operation **IXQuery::ClearExternalVariableList**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			
documentation	Clears the external variables list created with the AddExternalVariable method.					

Operation **IXQuery::DotNetExtensionsEnabled**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			
documentation	Enables or disables the Visual Studio .NET extensions. Properties: Returns a boolean value 'true' enables the .NET extensions, 'false' disables them.					

Operation **IXQuery::EngineVersion**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMXQueryVersion**			

documentation	Sets the XQuery version you are going to use (XQuery 1.0/3.0). Properties: The version number set by the EnumQueryVersion enumeration literal: eVersion10 or eVersion30.

Operation **IXQuery::Execute**

parameter	name	direction	type	type modifier	multiplicity	default
	bstrOutputFile	**in**	**string**			
	return	**return**	**bool**			
documentation	Executes the XQuery transformation (depending on the value of ENUMQueryVersion) and saves the result to an output file. Properties: 'bstrOutputFile' is the path (and file name) of the result file. If an error occurs during execution, use the LastErrorMessage operation to access additional information. Raises a RaptorXMLException when an error occurs.					

Operation **IXQuery::ExecuteAndGetResultAsString**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			
documentation	Executes the XQuery and returns the result as a string. Properties: Returns the transformation result as a string. If an error occurs during execution, use the getLastErrorMessage operation to access additional information. Raises a RaptorXMLException when an error occurs.					

Operation **IXQuery::IndentCharacters**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			
documentation	Sets the character string that will be used as indentation. Returns the indentation character - of type string.					

Operation **IXQuery::InputXMLFileName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			
documentation	Sets the file name (as a URL) of the input XML instance to be transformed. Properties: The name/URL of the XML file - of type string.					

	Note: Use the absolute URL for the input file name. An absolute URL specifies the exact location of the file, e.g. http://www.myWebsite.com/xmlfiles/myInputxml.xml

Operation **IXQuery::InputXMLFromText**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			
documentation	Supplies the contents of the XML input document as text. Properties: The input XML data - of type string. Note: Use the absolute URL for the input file name. An absolute URL specifies the exact location of the file, e.g. http://www.myWebsite.com/xmlfiles/myInputxml.xml					

Operation **IXQuery::IsValid**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			
documentation	Result of the XQuery 1.0/3.0 validation (depending on the value of ENUMQueryVersion). Properties: Returns a boolean. 'true' on success, 'false' on failure. Raises a RaptorXMLException when an error occurs.					

Operation **IXQuery::JavaBarcodeExtensionLocation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			
documentation	Enables or disables the Java Barcode extensions. Properties: Returns a boolean value. 'true' enables the Java Barcode extensions, 'false' disables them.					

Operation **IXQuery::JavaExtensionsEnabled**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			
documentation	Enables or disables the Java extensions. Properties: Returns a boolean value.					

	'true' enables the Java extensions, 'false' disables them.

Operation **IXQuery::LastErrorMessage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			
documentation	Retrieves the last error message from the XML engine. Properties: Returns the text of the last error message - of type string.					

Operation **IXQuery::LoadXMLWithPSVI**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			
documentation	Enables or disables the option 'Load post schema validation ínfoset'. Returns the boolean value. Values: 'true' enables the post schema validation infoset, 'false' disables it.					

Operation **IXQuery::OutputEncoding**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			
documentation	Sets the encoding for the result document. Properties: The encoding name (e.g. : UTF-8, UTF-16, ASCII, 8859-1, 1252) - of type string.					

Operation **IXQuery::OutputIndent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			
documentation	Enable/disables the indentation option for the result document. Properties: Returns a boolean value. 'true' enables indentation, 'false' disables it.					

Operation **IXQuery::OutputMethod**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			
documentation	Sets the serialization method for the result document. Properties: The serialization method - of type string. Values:					

	xml, xhtml, html, text.

Operation **IXQuery::OutputOmitXMLDeclaration**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			
documentation	Enables/disables the output of the XML declaration for the result document. Properties: Returns a boolean value. 'true' omits the XML declaration, 'false' includes it.					

Operation **IXQuery::XincludeSupport**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			
documentation	Enables or disables the use of XML Inclusions (XInclude elements). Returns a boolean value. Values: 'true' enables the XInclude include elements, 'false', ignores them.					

Operation **IXQuery::XMLValidationMode**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMXMLValidationMode**			
documentation	Sets the XML validation mode, depends on ENUMXMLValidationMode. Returns the value of the selected enumeration literal - of type ENUMXMLValidationMode.					

Operation **IXQuery::XQueryFileName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			
documentation	Sets the file name of the XQuery document. Properties: The absolute URL giving the base location of the XQuery file - of type string. An absolute URL specifies the exact location of the file, e.g. http://www.myWebsite.com/xmlfiles/myInputxml.xml					

Operation **IXQuery::XQueryFromText**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			
documen	Supplies the contents of the XQuery statement as text.					

tation	Properties: The XQuery statement - of type string.

Operation **IXQuery::XSDVersion**

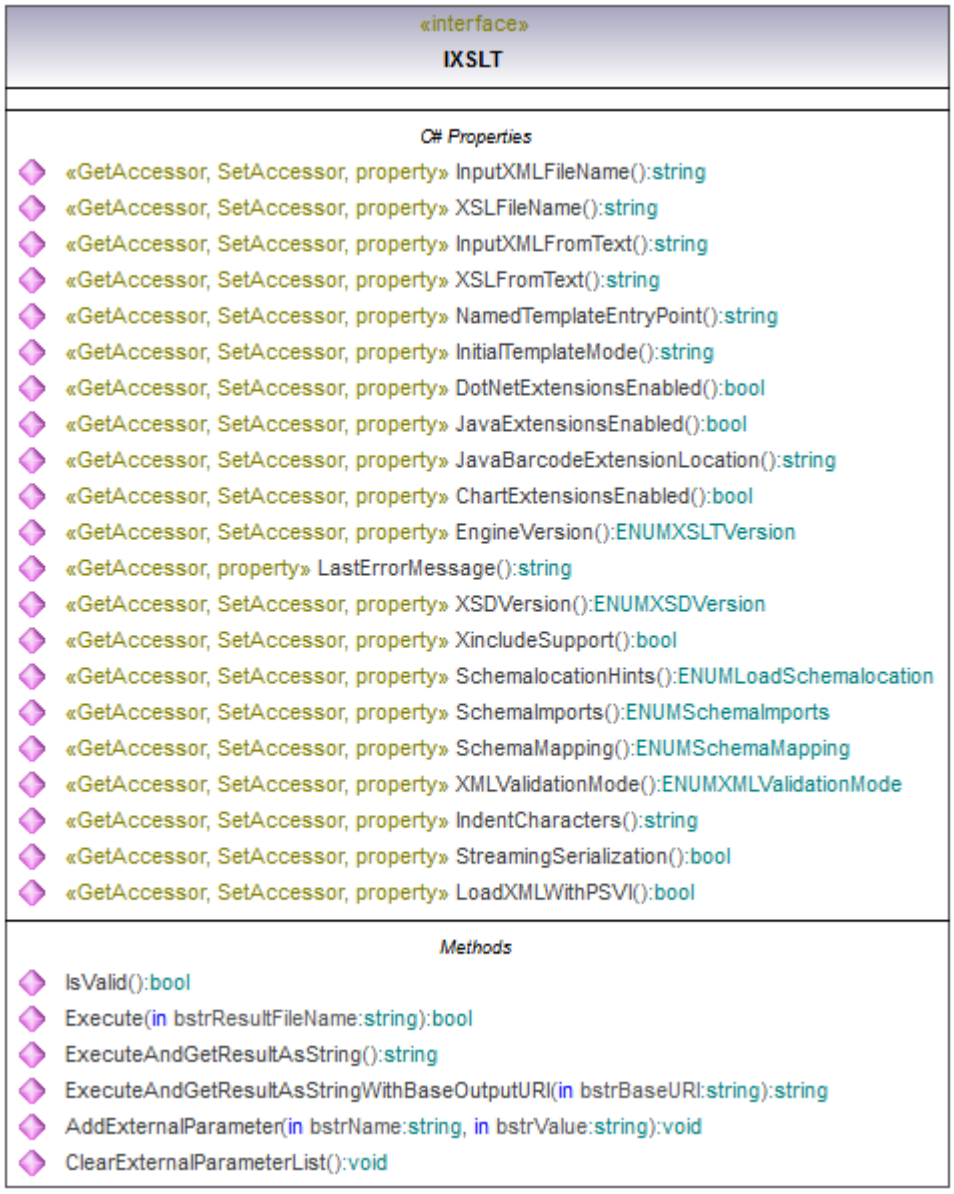
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMXSDVersion**			
documentation	Defines the XML Schema version that the document will be validated against. Returns the XML Schema version set by the EnumXSDVersion literal - of type EnumXSDVersion.					

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 06/07/13 15:48:10

5.16 RaptorXMLDev_COM - IXSLT

Interface **IXSLT**

diagram



documentation

The IXSLT object provides methods and properties to execute an XSLT 1.0/2.0/3.0 transformation using the RaptorXML 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 RaptorXML 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.

Operation **IXSLT::AddExternalParameter**

parameter	name	direction	type	type modifier	multiplicity	default
	bstrName	**in**	**string**			
	bstrValue	**in**	**string**			
	return	**return**	**void**			
documentation	Adds the name and value of a new external parameter. Properties: 'bstrName' is the parameter name, and is a valid QName - of type string. 'bstrValue' is the value of the parameter - of type string. Each external parameter and its value is to be specified in a separate call to the method. Parameters must be declared in the XQuery document, optionally with a type declaration. Whatever the type declaration for the external parameter in the XQuery document, the parameter value submitted to the AddExternalVariable does not need any special delimiter, such as quotes.					

Operation **IXSLT::ChartExtensionsEnabled**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			
documentation	Enables or disables the Altova Chart extensions. Properties: Returns a boolean value. 'true' enables the chart extensions, 'false' disables them.					

Operation **IXSLT::ClearExternalParameterList**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			
documentation	Clears the external parameters list created with the AddExternalParameter method.					

Operation **IXSLT::DotNetExtensionsEnabled**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			
documentation	Enables or disables the Visual Studio .NET extensions. Properties:					

	Returns a boolean value 'true' enables the .NET extensions, 'false' disables them.

Operation **IXSLT::EngineVersion**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMXSLTVersion**			
documentation	Sets the XSLT version you are going to use (XSLT1.0/2.0/3.0). Properties: The version number set by the EnumXSLTVersion enumeration literal: eVersion10, eVersion20, or eVersion30.					

Operation **IXSLT::Execute**

parameter	name	direction	type	type modifier	multiplicity	default
	bstrResultFileName	**in**	**string**			
	return	**return**	**bool**			
documentation	Executes the XSLT transformation (depending on the value of ENUMXSLTVersion) and saves the result to an output file. Properties: 'bstrResultFile' is the path (and file name) of the result file. If an error occurs during execution, use the LastErrorMessage operation to access additional information. Raises a RaptorXMLException when an error occurs.					

Operation **IXSLT::ExecuteAndGetResultAsString**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			
documentation	Executes the XSLT and returns the result as a string. Properties: Returns the transformation result as a string. If an error occurs during execution, use the LastErrorMessage operation to access additional information. Raises a RaptorXMLException when an error occurs.					

Operation **IXSLT::ExecuteAndGetResultAsStringWithBaseOutputURI**

parameter	name	direction	type	type modifier	multiplicity	default
	bstrBaseURI	**in**	**string**			
	return	**return**	**string**			

documentation	Executes the XSLT and returns the result as a string at the location definied by the base URI. Parameters: none. If an error occurs during execution, use the getLastErrorMessage operation to access additional information. Raises a RaptorXMLException when an error occurs.

Operation **IXSLT::IndentCharacters**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			
documentation	Sets the character string that will be used as indentation. 'Returns the indentation character - of type string.					

Operation **IXSLT::InitialTemplateMode**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			
documentation	Sets the initial mode for XSLT processing. Properties: The name of the required initial mode - of type string. Templates with this mode value will be processed. For example: SetInitialTemplateMode="MyMode". Note: Transformation must always occur after assigning the XML and XSLT documents.					

Operation **IXSLT::InputXMLFileName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			
documentation	Sets the file name (as a URL) of the input XML instance to be transformed. Properties: The name/URL of the XML file - of type string. Note: Use the absolute URL for the input file name. An absolute URL specifies the exact location of the file, e.g. http://www.myWebsite.com/xmlfiles/myInputxml.xml					

Operation **IXSLT::InputXMLFromText**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			
documentation	Supplies the contents of the XML input document as text.					

	Properties: The input XML data - of type string. Note: Use the absolute URL for the input file name. An absolute URL specifies the exact location of the file, e.g. http://www.myWebsite.com/xmlfiles/myInputxml.xml

Operation **IXSLT::IsValid**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			
documentation	Result of the XSLT validation (depends on the value of ENUMXSLTVersion) Properties: Returns a boolean. 'true' on success, 'false' on failure. Raises a RaptorXMLException when an error occurs.					

Operation **IXSLT::JavaBarcodeExtensionLocation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			
documentation	Defines the location of the Java Barcode extension file. Properties: The location of the extension file - of type string. Use the absolute URL for the input file name. An absolute URL specifies the exact location of the file e.g. e.g. http://www.myWebsite.com/xmlfiles/myInputxml.xml					

Operation **IXSLT::JavaExtensionsEnabled**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			
documentation	Enables or disables the Java extensions. Properties: Returns a boolean value. 'true' enables the Java extensions, 'false' disables them.					

Operation **IXSLT::LastErrorMessage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			
documentation	Retrieves the last error message from the XSLT engine. Properties: Returns the text of the last error message - of type string.					

Operation **IXSLT::LoadXMLWithPSVI**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			
documentation	Enables or disables the option 'Load post schema validation ínfoset'. Returns a boolean value. 'true' enables the post schema validation infoset, 'false' disables it.					

Operation **IXSLT::NamedTemplateEntryPoint**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			
documentation	Sets the named template entry point. Properties: The name of the node from which processing is to start - of type string.					

Operation **IXSLT::SchemaImports**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMSchemaImports**			
documentation	Defines the behaviour of xs:import elements, which each have an optional namespace attribute and an optional schemaLocation attribute, depends on ENUMSchemaImports.					

Operation **IXSLT::SchemalocationHints**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMLoadSchemalocation**			
documentation	Sets the location that the schema will be loaded from, depends on ENUMLoadSchemaLocation. Returns the value of the selected enumeration literal - of type ENUMLoadSchemaLocation.					

Operation **IXSLT::SchemaMapping**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMSchemaMapping**			
documentation	Sets the location that the schema will be loaded from, depends on ENUMSchemaMapping. Returns the value of the selected enumeration literal - of type ENUMSchemaMapping.					

Operation **IXSLT::StreamingSerialization**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			
documentation	Enables streaming serialization. In streaming mode, data stored in memory is minimized and processing is faster. Returns a boolean value -'true' enables streaming serialization, 'false' disables it. Default is true.					

Operation **IXSLT::XincludeSupport**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			
documentation	Enables or disables the use of XML Inclusions (XInclude elements). Returns a boolean value. 'true' enables the XInclude include elements, 'false', ignores them.					

Operation **IXSLT::XMLValidationMode**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMXMLValidationMode**			
documentation	Sets the XML validation mode, depends on ENUMXMLValidationMode. Returns the value of the selected enumeration literal - of type ENUMXMLValidationMode.					

Operation **IXSLT::XSDVersion**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMXSDVersion**			
documentation	Defines the XML Schema version that the document will be validated against. Returns the XML Schema version set by the EnumXSDVersion literal - of type EnumXSDVersion.					

Operation **IXSLT::XSLFileName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			
documentation	Sets the path/URL to locate the XSLT file to be used for the transformation. Properties: The path/file name of the XSLT file - of type string. Note: Use the absolute URL for the input file name. An absolute URL specifies the exact location of the file, e.g. http://www.myWebsite.com/xmlfiles/myInputxml.xml.					

Operation **IXSLT::XSLFromText**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			
documentation	Sets the XSLT file name to be used for the transformation. Properties: The XML text file name of the XSLT file - of type string.					

UML documentation generated by **UModel** UML Editor **http://www.altova.com/umodel** 06/07/13 15:48:10

Chapter 6

XSLT and XQuery Engine Information

6 XSLT and XQuery Engine Information

The XSLT and XQuery engines of RaptorXML follow the W3C specifications closely and are therefore stricter than previous Altova engines—such as those of AltovaXML, the predecessor of RaptorXML. As a result, minor errors that were ignored by previous engines, are flagged as errors by RaptorXML

For example:

- It is a type error (`err:XPTY0018`) if the result of a path operator contains both nodes and non-nodes.
- It is a type error (`err:XPTY0019`) if `E1` in a path expression `E1/E2` does not evaluate to a sequence of nodes.

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

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

6.1 XSLT 1.0

The XSLT 1.0 Engine of RaptorXML (all editions) conforms to the World Wide Web Consortium's (W3C's) XSLT 1.0 Recommendation of 16 November 1999 and XPath 1.0 Recommendation of 16 November 1999. Note the following information about the implementation.

Notes about the implementation

When the `method` attribute of `xsl:output` is set to HTML, or if HTML output is selected by default, then special characters in the XML or XSLT file are inserted in the HTML document as HTML character references in the output. For instance, the character ` ` (the decimal character reference for a non-breaking space) is inserted as ` ` in the HTML code.

6.2 XSLT 2.0

This section:

Conformance
The XSLT 2.0 engine of RaptorXML (all editions) conforms to the World Wide Web Consortium's (W3C's) XSLT 2.0 Recommendation of 23 January 2007 and XPath 2.0 Recommendation of 14 December 2010.

Backwards Compatibility
The XSLT 2.0 engine is backwards compatible. The only time the backwards compatibility of the XSLT 2.0 engine comes into effect is when using the XSLT 2.0 engine (CLI parameter `--xslt=2`) to process an XSLT 1.0 stylesheet. Note that there could be differences in the outputs produced by the XSLT 1.0 Engine and the backwards-compatible XSLT 2.0 engine.

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

Namespace Name	Prefix	Namespace URI
XML Schema types	`xs:`	`http://www.w3.org/2001/XMLSchema`
XPath 2.0 functions	`fn:`	`http://www.w3.org/2005/xpath-functions`

Typically, these namespaces will be declared on the `xsl:stylesheet` or `xsl:transform` element, as shown in the following listing:

```
<xsl:stylesheet version="2.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:fn="<%NS-FN%>"
    ...
</xsl:stylesheet>
```

The following points should be noted:

- The XSLT 2.0 engine uses the XPath 2.0 and XQuery 1.0 Functions namespace (listed in the table above) as its **default functions namespace**. So you can use XPath 2.0 and XSLT 2.0 functions in your stylesheet without any prefix. If you declare the XPath

2.0 Functions namespace in your stylesheet with a prefix, then you can additionally use the prefix assigned in the declaration.

- When using type constructors and types from the XML Schema namespace, the prefix used in the namespace declaration must be used when calling the type constructor (for example, `xs:date`).
- Some XPath 2.0 functions have the same name as XML Schema datatypes. For example, for the XPath functions `fn:string` and `fn:boolean` there exist XML Schema datatypes with the same local names: `xs:string` and `xs:boolean`. So if you were to use the XPath expression `string('Hello')`, the expression evaluates as `fn:string('Hello')`—not as `xs:string('Hello')`.

Schema-awareness
The XSLT 2.0 engine is schema-aware. So you can use user-defined schema types and the `xsl:validate` instruction.

Implementation-specific behavior
Given below is a description of how the XSLT 2.0 engine handles implementation-specific aspects of the behavior of certain XSLT 2.0 functions.

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

`function-available`
The function tests for the availability of in-scope functions (XSLT 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 (the Altova-specific): `x-binarytobase16` and `x-binarytobase64`.

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

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

XPath 2.0 functions
Implementation-specific behavior of XPath 2.0 functions is listed in the section XQuery 1.0 and XPath 2.0 Functions.

6.3 XSLT 3.0

The XSLT 3.0 Engine of RaptorXML (all editions) conforms to the World Wide Web Consortium's (W3C's) XSLT 3.0 Working Draft of 10 July 2012 and XPath 3.0 Recommendation of 8 January 2013.

The XSLT 3.0 engine has the same implementation-specific characteristics as the XSLT 2.0 engine. Additionally, it supports the following XSLT 3.0 features: `xsl:evaluate`, `xsl:try`, `xsl:catch`, XPath and XQuery 3.0 functions and operators, and the XPath 3.0 specification.

6.4 XQuery 1.0

This section:

Conformance
The XQuery 1.0 Engine of RaptorXML (all editions) conforms to the World Wide Web Consortium's (W3C's) XQuery 1.0 Recommendation of 14 December 2010. The XQuery standard gives implementations discretion about how to implement many features. Given below is a list explaining how the XQuery 1.0 Engine implements these features.

Schema awareness
The XQuery 1.0 Engine is **schema-aware**.

Encoding
The UTF-8 and UTF-16 character encodings are supported.

Namespaces
The following namespace URIs and their associated bindings are pre-defined.

Namespace Name	Prefix	Namespace URI
XML Schema types	`xs:`	`http://www.w3.org/2001/XMLSchema`
Schema instance	`xsi:`	`http://www.w3.org/2001/XMLSchema-instance`
Built-in functions	`fn:`	`http://www.w3.org/2005/xpath-functions`
Local functions	`local:`	`http://www.w3.org/2005/xquery-local-functions`

The following points should be noted:

- The XQuery 1.0 Engine recognizes the prefixes listed above as being bound to the corresponding namespaces.
- Since the built-in functions namespace listed above is the default functions namespace in XQuery, the `fn:` prefix does not need to be used when built-in functions are invoked (for example, `string("Hello")` will call the `fn:string` function). However, the prefix `fn:` can be used to call a built-in function without having to declare the namespace in the query prolog (for example: `fn:string("Hello")`).
- You can change the default functions namespace by declaring the `default function namespace` expression in the query prolog.
- When using types from the XML Schema namespace, the prefix `xs:` may be used without having to explicitly declare the namespaces and bind these prefixes to them in the query prolog. (Example: `xs:date` and `xs:yearMonthDuration`.) If you wish to use some other prefix for the XML Schema namespace, this must be explicitly declared in the query prolog. (Example: `declare namespace alt = "http://www.w3.org/2001/XMLSchema"; alt:date("2004-10-04")`.)
- Note that the `untypedAtomic`, `dayTimeDuration`, and `yearMonthDuration` datatypes have been moved, with the CRs of <%CR-XQ1-DATE%>, from the XPath Datatypes namespace to the XML Schema namespace, so: `xs:yearMonthDuration`.

If namespaces for functions, type constructors, node tests, etc are wrongly assigned, an error is reported. Note, however, that some functions have the same name as schema datatypes, e.g. `fn:string` and `fn:boolean`. (Both `xs:string` and `xs:boolean` are defined.) The namespace prefix determines whether the function or type constructor is used.

XML source document and validation
XML documents used in executing an XQuery document with the XQuery 1.0 Engine must be well-formed. However, they do not need to be valid according to an XML Schema. If the file is not valid, the invalid file is loaded without schema information. If the XML file is associated with an external schema and is valid according to it, then post-schema validation information is generated for the XML data and will be used for query evaluation.

Static and dynamic type checking
The static analysis phase checks aspects of the query such as syntax, whether external references (e.g. for modules) exist, whether invoked functions and variables are defined, and so on. If an error is detected in the static analysis phase, it is reported and the execution is stopped.

Dynamic type checking is carried out at run-time, when the query is actually executed. If a type is incompatible with the requirement of an operation, an error is reported. For example, the expression `xs:string("1") + 1` returns an error because the addition operation cannot be carried out on an operand of type `xs:string`.

Library Modules
Library modules store functions and variables so they can be reused. The XQuery 1.0 Engine supports modules that are stored in **a single external XQuery file**. Such a module file must

contain a `module` declaration in its prolog, which associates a target namespace. Here is an example module:

```
module namespace libns="urn:module-library";
declare variable $libns:company := "Altova";
declare function libns:webaddress() { "http://www.altova.com" };
```

All functions and variables declared in the module belong to the namespace associated with the module. The module is used by importing it into an XQuery file with the `import module` statement in the query prolog. The `import module` statement only imports functions and variables declared directly in the library module file. As follows:

```
import module namespace modlib = "urn:module-library" at
   "modulefilename.xq";
if        ($modlib:company = "Altova")
then       modlib:webaddress()
else       error("No match found.")
```

External functions
External functions are not supported, i.e. in those expressions using the `external` keyword, as in:

```
declare function hoo($param as xs:integer) as xs:string external;
```

Collations
The default collation is the Unicode-codepoint collation, which compares strings on the basis of their Unicode codepoint. Other supported collations are the ICU collations listed here. To use a specific collation, supply its URI as given in the list of supported collations. Any string comparisons, including for the `fn:max` and `fn:min` functions, will be made according to the specified collation. If the collation option is not specified, the default Unicode-codepoint collation is used.

Precision of numeric types

- The `xs:integer` datatype is arbitrary-precision, i.e. it can represent any number of digits.
- The `xs:decimal` datatype has a limit of 20 digits after the decimal point.
- The `xs:float` and `xs:double` datatypes have limited-precision of 15 digits.

XQuery Instructions Support
The `Pragma` instruction is not supported. If encountered, it is ignored and the fallback expression is evaluated.

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.

6.5 XQuery 3.0

The XQuery 3.0 Engine of RaptorXML (all editions) conforms to the World Wide Web Consortium's (W3C's) XQuery 3.0 Recommendation of 8 January 2013 and includes support for XPath and XQuery Functions 3.0.

Implementation-specific characteristics are the same as for XQuery 1.0.

6.6 XQuery 1.0 and XPath 2.0 Functions

This section:

Conformance
The XQuery 1.0 and XPath 2.0 functions support of RaptorXML (all editions) conforms to the World Wide Web Consortium's (W3C's) XQuery 1.0 and XPath 2.0 Functions and Operators Recommendation of 14 December 2010.

General points
Given below is a list (in alphabetical order) of the implementation-specific behavior of certain functions. The following general points should be noted:

- The default functions namespace has been set to comply with that specified in the standard. Functions can therefore be called without a prefix.
- In general, if a function expects a sequence of one item as an argument, and a sequence of more than one item is submitted, then an error is returned.
- All string comparisons are done using the Unicode codepoint collation.
- Results that are QNames are serialized in the form `[prefix:]localname`.

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

Implicit timezone
When two `date`, `time`, or `dateTime` values need to be compared, the 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
The default collation is the Unicode codepoint collation, which compares strings on the basis of

their Unicode codepoint. Other supported collations are the ICU collations listed below. To use a specific collation, supply its URI as given in the list of supported collations. Any string comparisons, including for the `fn:max` and `fn:min` functions, will be made according to the specified collation. If the collation option is not specified, the default Unicode-codepoint collation is used.

Language	URIs
`da`: Danish	`da_DK`
`de`: German	`de_AT, de_BE, de_CH, de_DE, de_LI, de_LU`
`en`: English	`en_AS, en_AU, en_BB, en_BE, en_BM, en_BW, en_BZ, en_CA, en_GB, en_GU, en_HK, en_IE, en_IN, en_JM, en_MH, en_MP, en_MT, en_MU, en_NA, en_NZ, en_PH, en_PK, en_SG, en_TT, en_UM, en_US, en_VI, en_ZA, en_ZW`
`es`: Spanish	`es_419, es_AR, es_BO, es_CL, es_CO, es_CR, es_DO, es_EC, es_ES, es_GQ, es_GT, es_HN, es_MX, es_NI, es_PA, es_PE, es_PR, es_PY, es_SV, es_US, es_UY, es_VE`
`fr`: French	`fr_BE, fr_BF, fr_BI, fr_BJ, fr_BL, fr_CA, fr_CD, fr_CF, fr_CG, fr_CH, fr_CI, fr_CM, fr_DJ, fr_FR, fr_GA, fr_GN, fr_GP, fr_GQ, fr_KM, fr_LU, fr_MC, fr_MF, fr_MG, fr_ML, fr_MQ, fr_NE, fr_RE, fr_RW, fr_SN, fr_TD, fr_TG`
`it`: Italian	`it_CH, it_IT`
`ja`: Japanese	`ja_JP`
`nb`: Norwegian Bokmal	`nb_NO`
`nl`: Dutch	`nl_AW, nl_BE, nl_NL`
`nn`: Nynorsk	`nn_NO`
`pt`: Portuguese	`pt_AO, pt_BR, pt_GW, pt_MZ, pt_PT, pt_ST`
`ru`: Russian	`ru_MD, ru_RU, ru_UA`
`sv`: Swedish	`sv_FI, sv_SE`

Namespace axis
The namespace axis is deprecated in XPath 2.0. Use of the namespace axis is, however, supported. To access namespace information with XPath 2.0 mechanisms, use the `fn:in-scope-prefixes()`, `fn:namespace-uri()` and `fn:namespace-uri-for-prefix()` functions.

`base-uri`

- If external entities are used in the source XML document and if a node in the external entity is specified as the input node argument of the `base-uri()` function, it is still the base URI of the including XML document that is used—not the base URI of the external entity.
- The base URI of a node in the XML document can be modified using the `xml:base`

attribute.

---

collection

- The argument is a relative URI that is resolved against the current base URI.
- If the resolved URI identifies an XML file, then this XML file is treated as a catalog which references a collection of files. This file must have the form:

```
<collection>
    <doc href="uri-1" />
    <doc href="uri-2" />
    <doc href="uri-3" />
 </collection>
```

  The files referenced by the href attributes are loaded, and their document nodes are returned as a sequence.
- If the resolved URI does not identify an XML file with the catalog structure described above, then the argument string (in which wildcards such as `?` and `*` are allowed) is used as a search string. XML files with names that match the search expression are loaded, and their document nodes are returned as a sequence. See examples below.
- XSLT example: The expression `collection("c:\MyDocs\*.xml")//Title` returns a sequence of all `DocTitle` elements in the `.xml` files in the `MyDocs` folder.
- XQuery example: The expression `{for $i in collection(c:\MyDocs\*.xml) return element doc{base-uri($i)}}` returns the base URIs of all the `.xml` files in the `MyDocs` folder, each URI being within a `doc` element.
- The default collection is empty.

---

current-date, current-dateTime, current-time

- The current date and time is taken from the system clock.
- The timezone is taken from the implicit timezone provided by the evaluation context; the implicit timezone is taken from the system clock.
- The timezone is always specified in the result.

---

doc

An error is raised only if no XML file is available at the specified location or if the file is not well-formed. The file is validated if a schema is available. If the file is not valid, the invalid file is loaded without schema information.

---

id

In a well-formed but invalid document that contains two or more elements having the same ID value, the first element in document order is returned.

---

in-scope-prefixes

Only default namespaces may be undeclared in the XML document. However, even when a

default namespace is undeclared on an element node, the prefix for the default namespace, which is the zero-length string, is returned for that node.

---

normalize-unicode

The normalization forms NFC, NFD, NFKC, and NFKD are supported.

---

resolve-uri

- If the second, optional argument is omitted, the URI to be resolved (the first argument) is resolved against the base URI from the static context, which is the URI of the XSLT stylesheet or the base URI given in the prolog of the XQuery document.
- The relative URI (the first argument) is appended after the last "/" in the path notation of the base URI notation.
- If the value of the first argument is the zero-length string, the base URI from the static context is returned, and this URI includes the file name of the document from which the base URI of the static context is derived (e.g. the XSLT or XML file).

---

static-base-uri

The base URI from the static context is the base URI of the XSLT stylesheet or the base URI specified in the prolog of the XQuery document.

6.7 XPath and XQuery Functions 3.0

The XPath and XQuery 3.0 functions and operators support of RaptorXML (all editions) conforms to the World Wide Web Consortium's (W3C's) XPath and XQuery Functions and Operators 3.0 Recommendation of 21 May 2013.

Implementation-specific characteristics are the same as for XQuery 1.0 and XPath 2.0 Functions.

Chapter 7

XSLT and XQuery Extension Functions

7 XSLT and XQuery Extension Functions

There are several ready-made functions in programming languages such as Java and C# that are not available as XQuery/XPath functions or as XSLT functions. A good example would be the math functions available in Java, such as `sin()` and `cos()`. If these functions were available to the designers of XSLT stylesheets and XQuery queries, it would increase the application area of stylesheets and queries and greatly simplify the tasks of stylesheet creators.

The XSLT and XQuery engines used in a number of Altova products support the use of extension functions in Java and .NET. They also support MSXSL scripts for XSLT and Altova's own extension functions.

You should note that, with the exception of some Altova extension functions for XSLT, nearly all of the extension functions in this section are called from XPath expressions. This section describes how to use extension functions and MSXSL scripts in your XSLT stylesheets and XQuery documents. The available extension functions are organized into the following sections:

- Altova Extension Functions
- Java Extension Functions
- .NET Extension Functions
- MSXSL Scripts for XSLT

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.

7.1 Altova Extension Functions

Altova extension functions are in the **Altova extension functions namespace**

```
http://www.altova.com/xslt-extensions
```

and are indicated in this section with the prefix

```
altova:
```

which is assumed to be bound to the namespace given above.

The following extension functions are supported in the current version of your Altova product in the manner described below.

General functions

XPath functions

These functions can be used in XPath contexts:

- altova:generate-auto-number()
- altova:reset-auto-number()
- altova:get-temp-folder()

XSLT functions

These functions can be used in an XSLT context, just like XSLT 2.0's `current-group()` or `key()` functions:

- altova:evaluate()
- altova:distinct-nodes()
- altova:encode-for-rtf()
- altova:xbrl-labels()
- altova:xbrl-footnotes()

7.1.1 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.

Note that Altova extension functions are in the **Altova extension functions namespace**

```
http://www.altova.com/xslt-extensions
```

and are indicated in this section with the prefix

```
altova:
```

which is assumed to be bound to the namespace given above.

Functions for use in XPath contexts
These functions can be used in XPath contexts:

- altova:generate-auto-number()
- altova:reset-auto-number()
- altova:get-temp-folder()

Functions for use in XSLT contexts
These functions can be used in an XSLT context, similarly to the way in which XSLT 2.0's `current-group()` or `key()` functions are used:

- altova:evaluate()
- altova:distinct-nodes()
- altova:encode-for-rtf()
- altova:xbrl-labels()
- altova:xbrl-footnotes()

Functions for use in XPath contexts
These functions can be used in XPath contexts:

altova:generate-auto-number(id as xs:string, start-with as xs:double, increment as xs:double, 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.

Functions for use in XSLT contexts
These functions can be used in an XSLT context, just like XSLT 2.0's `current-group()` or `key()` functions:

`altova:evaluate()`
The `altova:evaluate()` function takes an XPath expression, passed as a string, as its mandatory argument. It returns the output of the evaluated expression.

```
altova:evaluate(XPathExp as xs:string)
```

For example:

```
altova:evaluate('//Name[1]')
```

In the example above, note that the expression `//Name[1]` is passed as a string by enclosing it in single quotes. The `altova:evaluate` function returns the contents of the first `Name` element in the document.

The `altova:evaluate` function can take additional (optional) arguments. These arguments are, respectively, the values of variables with the names `p1`, `p2`, `p3`... `pN` that can be used in the XPath expression.

```
altova:evaluate(XPathExp as xs:string [, p1value ... pNvalue])
```

where

- the variable names must be of the form `pX`, `X` being an integer
- the sequence of the function's arguments, from the second argument onwards corresponds to the sequence of variables named `p1` to `pN`. So the second argument will be the value of the variable `p1`, the third argument that of the variable `p2`, and so on.
- The variable values must be of type `item*`

For example:

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

In the above listing, notice the following:

- The second argument of the `altova:evaluate` expression is the value assigned to the variable `$p1`, the third argument that assigned to the variable `$p2`, and so on.

- Notice that the fourth argument of the function is a string value, indicated by its being enclosed in quotes.
- The `select` attribute of the `xs:variable` element supplies the XPath expression. Since this expression must be of type `xs:string`, it is enclosed in single quotes.

The following examples further illustrate usage:

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

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

The `altova:evaluate()` extension function is useful in situations where an XPath expression in the XSLT stylesheet contains one or more parts that must be evaluated dynamically. For example, consider a situation in which a user enters his request for the sorting criterion and this criterion is stored in the attribute `UserReq/@sortkey`. In the stylesheet, you could then have the expression :

```
<xsl:sort select="altova:evaluate(../UserReq/@sortkey)" order="ascending"/
>
```

The `altova:evaluate()` function reads the `sortkey` attribute of the `UserReq` child element of the parent of the context node. Say the value of the `sortkey` attribute is `Price`, then `Price` is returned by the `altova:evaluate()` function and becomes the value of the `select` attribute:

```
<xsl:sort select="Price" order="ascending"/>
```

If this `sort` instruction occurs within the context of an element called `Order`, then the `Order` elements will be sorted according to the values of their `Price` children. Alternatively, if the value of `@sortkey` were, say, `Date`, then the `Order` elements would be sorted according to the values of their `Date` children. So the sort criterion for `Order` is selected from the `sortkey` attribute at runtime. This could not have been achieved with an expression like:

```
<xsl:sort select="../UserReq/@sortkey" order="ascending"/>
```

In the case shown above, the sort criterion would be the `sortkey` attribute itself, not `Price` or `Date` (or any other current content of `sortkey`).

Variables can be used in the `altova:evaluate()` extension function as shown in the examples below:

- Static variables: `<xsl:value-of select="$i3, $i2, $i1" />`
  Outputs the values of three variables.

- Dynamic XPath expression with dynamic variables:
  ```
  <xsl:variable name="xpath" select="'$p3, $p2, $p1'" />
  <xsl:value-of select="altova:evaluate( $xpath, 10, 20, 30 )" />
  ```
  Outputs "`30 20 10`*"*

- Dynamic XPath expression with no dynamic variable:
  ```
  <xsl:variable name="xpath" select="'$p3, $p2, $p1'" />
  <xsl:value-of select="altova:evaluate( $xpath )" />
  ```
  Outputs error: No variable defined for $p3.

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

altova:distinct-nodes()

The `altova:distinct-nodes()` function takes a set of one or more nodes as its input and returns the same set minus nodes with duplicate values. The comparison is done using the XPath/XQuery function `fn:deep-equal`.

```
altova:distinct-nodes( $arg as node()* )  as node()*
```

altova:encode-for-rtf()

The `altova:encode-for-rtf()` function converts the input string into code for RTF.

```
altova:encode-for-rtf( $inputstr as xs:string?,
$preserveallwhitespace as xs:boolean,
$preservenewlines as xs:boolean) as xs:string
```

Whitespace and new lines will be preserved according to the boolean value specified for their respective parameters.

altova:xbrl-labels()

The `altova:xbrl-labels()` function takes two input arguments: a node name and the taxonomy file location containing the node. The function returns the XBRL labels associated with the input node.

```
altova:xbrl-labels( $name as xs:QName, $file as xs:string )  as node()*
```

altova:xbrl-footnotes()

The `altova:footnotes()` function takes a node as its input argument and returns the set of XBRL footnote nodes referenced by the input node.

```
altova:footnotes( $arg as node() )  as node()*
```

7.1.2 Barcode Functions

The XSLT Engine uses 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 *<ProgramFilesFolder>*`\Altova\Common2013\jar`.

The Java libraries used are in sub-folders of the folder *<ProgramFilesFolder>*`\Altova\Common2013\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>
            <img alt="barcode" src="{altovaext:get-temp-folder()}barcode.png
"/>
         </body>
      </html>
      <xsl:result-document
         href="{altovaext:get-temp-folder()}barcode.png"
         method="text" encoding="base64tobinary" >
         <xsl:variable name="barcodeObject"
            select="
altovaext-barcode:newInstance(&apos;Code39&apos;,string(&apos;some
value&apos;),
            96,0, (altovaext-barcode-property:new( &apos;setModuleWidth&apos;,
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>
```

7.2 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:

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.

7.2.1 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. (*See example below*.)
- The class file is not packaged. The XSLT or XQuery file is in the same folder as the class file. (*See example below*.)
- The class file is in a package. The XSLT or XQuery file is at some random location. (*See example below*.)
- The class file is not packaged. The XSLT or XQuery file is at some random location. (*See example below*.)

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.

7.2.2 User-Defined Jar Files

If access is via a JAR file, the URI of the JAR file must be specified using the following syntax:

```
xmlns:classNS="java:classname?path=jar:uri-of-jarfile!/"
```

The method is then called by using the prefix of the namespace URI that identifies the class: `classNS:method()`

In the above:

`java:` indicates that a Java function is being called
`classname` is the name of the user-defined class
`?` is the separator between the classname and the path
`path=jar:` indicates that a path to a JAR file is being given
`uri-of-jarfile` is the URI of the jar file
`!/` is the end delimiter of the path
`classNS:method()` is the call to the method

Alternatively, the classname can be given with the method call. Here are two examples of the syntax:

```
xmlns:ns1="java:docx.layout.pages?path=jar:file:///c:/projects/docs/docx.jar!/
"
      ns1:main()

      xmlns:ns2="java?path=jar:file:///c:/projects/docs/docx.jar!/"
      ns2:docx.layout.pages.main()
```

Here is a complete XSLT example that uses a JAR file to call a Java extension function:

```
<xsl:stylesheet version="2.0"
      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:fn="http://www.w3.org/2005/xpath-functions"
      xmlns:car="java?path=jar:file:///C:/test/Car1.jar!/" >
<xsl:output exclude-result-prefixes="fn car xsl xs"/>

<xsl:template match="/">
      <xsl:variable name="myCar" select="car:Car1.new('red')" />
      <a><xsl:value-of select="car:Car1.getCarColor($myCar)"/></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.

7.2.3 Java: Constructors

An extension function can be used to call a Java constructor. All constructors are called with the pseudo-function `new()`.

If the result of a Java constructor call can be implicitly converted to XPath/XQuery datatypes, 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" />
```


7.2.4 Java: Static Methods and Static Fields

A static method is called directly by its Java name and by supplying the arguments for the method. Static fields (methods that take no arguments), such as the constant-value fields `E` and `PI`, are accessed without specifying any argument.

XSLT examples
Here are some examples of how static methods and fields can be called:

```
<xsl:value-of xmlns:jMath="java:java.lang.Math"
              select="jMath:cos(3.14)" />

<xsl:value-of xmlns:jMath="java:java.lang.Math"
              select="jMath:cos( jMath:PI() )" />

<xsl:value-of xmlns:jMath="java:java.lang.Math"
              select="jMath:E() * jMath:cos(3.14)" />
```

Notice that the extension functions above have the form `prefix:fname()`. The prefix in all three cases is `jMath:`, which is associated with the namespace URI `java:java.lang.Math`. (The namespace URI must begin with `java:`. In the examples above it is extended to contain the class name (`java.lang.Math`).) The `fname()` part of the extension functions must match the name of a public class (e.g. `java.lang.Math`) followed by the name of a public static method with its argument/s (such as `cos(3.14)`) or a public static field (such as `PI()`).

In the examples above, the class name has been included in the namespace URI. If it were not contained in the namespace URI, then it would have to be included in the `fname()` part of the extension function. For example:

```
<xsl:value-of xmlns:java="java:"
              select="java:java.lang.Math.cos(3.14)" />
```

XQuery example
A similar example in XQuery would be:

```
<cosine xmlns:jMath="java:java.lang.Math">
   {jMath:cos(3.14)}
</cosine>
```

7.2.5 Java: Instance Methods and Instance Fields

An instance method has a Java object passed to it as the first argument of the method call. Such a Java object typically would be created by using an extension function (for example a constructor call) or a stylesheet parameter/variable. An XSLT example of this kind would be:

```
<xsl:stylesheet version="1.0" exclude-result-prefixes="date"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:date="java:java.util.Date"
  xmlns:jlang="java:java.lang">
  <xsl:param name="CurrentDate" select="date:new()"/>
  <xsl:template match="/">
      <enrollment institution-id="Altova School"
                date="{date:toString($CurrentDate)}"
                type="{jlang:Object.toString(jlang:Object.getClass( date:new()
))}">
      </enrollment>
  </xsl:template>
</xsl:stylesheet>
```

In the example above, the value of the node `enrollment/@type` is created as follows:

1. An object is created with a constructor for the class `java.util.Date` (with the `date:new()` constructor).
2. This Java object is passed as the argument of the `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`.

7.2.6 Datatypes: XPath/XQuery to Java

When a Java function is called from within an XPath/XQuery expression, the datatype of the function's arguments is important in determining which of multiple Java classes having the same name is called.

In Java, the following rules are followed:

- If there is more than one Java method with the same name, but each has a different number of arguments than the other/s, then the Java method that best matches the number of arguments in the function call is selected.
- The XPath/XQuery string, number, and boolean datatypes (*see list below*) are implicitly converted to a corresponding Java datatype. If the supplied XPath/XQuery type can be converted to more than one Java type (for example, `xs:integer`), then that Java type is selected which is declared for the selected method. For example, if the Java method being called is `fx(decimal)` and the supplied XPath/XQuery datatype is `xs:integer`, then `xs:integer` will be converted to Java's `decimal` datatype.

The table below lists the implicit conversions of XPath/XQuery string, number, and boolean types to Java datatypes.

`xs:string`	`java.lang.String`
`xs:boolean`	`boolean` (primitive), `java.lang.Boolean`
`xs:integer`	int, long, short, byte, float, double, and the wrapper classes of these, such as `java.lang.Integer`
`xs:float`	`float` (primitive), `java.lang.Float`, `double` (primitive)
`xs:double`	`double` (primitive), `java.lang.Double`
`xs:decimal`	`float` (primitive), `java.lang.Float`, `double` (primitive), `java.lang.Double`

Subtypes of the XML Schema datatypes listed above (and which are used in XPath and XQuery) will also be converted to the Java type/s corresponding to that subtype's ancestor type.

In some cases, it might not be possible to select the correct Java method based on the supplied information. For example, consider the following case.

- The supplied argument is an `xs:untypedAtomic` value of 10 and it is intended for the method `mymethod(float)`.
- However, there is another method in the class which takes an argument of another datatype: `mymethod(double)`.
- Since the method names are the same and the supplied type (`xs:untypedAtomic`) could be converted correctly to either `float` or `double`, it is possible that `xs:untypedAtomic` is converted to `double` instead of `float`.
- Consequently the method selected will not be the required method and might not produce the expected result. To work around this, you can create a user-defined method with a different name and use this method.

Types that are not covered in the list above (for example `xs:date`) will not be converted and will generate an error. However, note that in some cases, it might be possible to create the required Java type by using a Java constructor.

7.2.7 Datatypes: Java to XPath/XQuery

When a Java method returns a value, the datatype of the value is a string, numeric or boolean type, then it is converted to the corresponding XPath/XQuery type. For example, Java's `java.lang.Boolean` and `boolean` datatypes are converted to `xsd:boolean`.

One-dimensional arrays returned by functions are expanded to a sequence. Multi-dimensional arrays will not be converted, and should therefore be wrapped.

When a wrapped Java object or a datatype other than string, numeric or boolean is returned, you can ensure conversion to the required XPath/XQuery type by first using a Java method (e.g `toString`) to convert the Java object to a string. In XPath/XQuery, the string can be modified to fit the lexical representation of the required type and then converted to the required type (for example, by using the `cast as` expression).

7.3 .NET Extension Functions

If you are working on the .NET platform on a Windows machine, you can use extension functions written in any of the .NET languages (for example, C#). A .NET extension function can be used within an XPath or XQuery expression to invoke a constructor, property, or method (static or instance) within a .NET class.

A property of a .NET class is called using the syntax `get_PropertyName()`.

This section is organized into the following sub-sections:

Form of the extension function
The extension function in the XPath/XQuery expression must have the form `prefix:fname()`.

- The `prefix:` part is associated with a URI that identifies the .NET class being addressed.
- The `fname()` part identifies the constructor, property, or method (static or instance) within the .NET class, and supplies any argument/s, if required.
- The URI must begin with `clitype:` (which identifies the function as being a .NET extension function).
- The `prefix:fname()` form of the extension function can be used with system classes and with classes in a loaded assembly. However, if a class needs to be loaded, additional parameters containing the required information will have to be supplied.

Parameters
To load an assembly, the following parameters are used:

`asm`	The name of the assembly to be loaded.
`ver`	The version number (maximum of four integers separated by periods).
`sn`	The key token of the assembly's strong name (16 hex digits).
`from`	A URI that gives the location of the assembly (DLL) to be loaded. If the URI is relative, it is relative to the XSLT or XQuery document. If this parameter is present, any other parameter is ignored.
`partialname`	The partial name of the assembly. It is supplied to `Assembly.LoadWith.PartialName()`, which will attempt to load the assembly. If `partialname` is present, any other parameter is ignored.
`loc`	The locale, for example, `en-US`. The default is `neutral`.

If the assembly is to be loaded from a DLL, use the `from` parameter and omit the `sn` parameter.

If the assembly is to be loaded from the Global Assembly Cache (GAC), use the `sn` parameter and omit the `from` parameter.

A question mark must be inserted before the first parameter, and parameters must be separated by a semi-colon. The parameter name gives its value with an equals sign (*see example below*).

Examples of namespace declarations
An example of a namespace declaration in XSLT that identifies the system class `System.Environment`:

```
xmlns:myns="clitype:System.Environment"
```

An example of a namespace declaration in XSLT that identifies the class to be loaded as `Trade.Forward.Scrip`:

```
xmlns:myns="clitype:Trade.Forward.Scrip?asm=forward;version=10.6.2.1"
```

An example of a namespace declaration in XQuery that identifies the system class `MyManagedDLL.testClass`:. Two cases are distinguished:

1. When the assembly is loaded from the GAC:
   ```
   declare namespace cs="clitype:MyManagedDLL.testClass?asm=MyManagedDLL;
   ver=1.2.3.4;loc=neutral;sn=b9f091b72dccfba8";
   ```

2. When the assembly is loaded from the DLL (complete and partial references below):
   ```
   declare namespace
cs="clitype:MyManagedDLL.testClass?from=file:///C:/Altova
   Projects/extFunctions/MyManagedDLL.dll;

   declare namespace
   cs="clitype:MyManagedDLL.testClass?from=MyManagedDLL.dll;
   ```

XSLT example
Here is a complete XSLT example that calls functions in system class `System.Math`:

```
<xsl:stylesheet version="2.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:fn="http://www.w3.org/2005/xpath-functions">
   <xsl:output method="xml" omit-xml-declaration="yes" />
   <xsl:template match="/">
      <math xmlns:math="clitype:System.Math">
         <sqrt><xsl:value-of select="math:Sqrt(9)"/></sqrt>
         <pi><xsl:value-of select="math:PI()"/></pi>
         <e><xsl:value-of select="math:E()"/></e>
         <pow><xsl:value-of select="math:Pow(math:PI(), math:E())"/></pow>
      </math>
   </xsl:template>
</xsl:stylesheet>
```

The namespace declaration on the element `math` associates the prefix `math:` with the URI `clitype:System.Math`. The `clitype:` beginning of the URI indicates that what follows identifies either a system class or a loaded class. The `math:` prefix in the XPath expressions associates the extension functions with the URI (and, by extension, the class) `System.Math`.

The extension functions identify methods in the class `System.Math` and supply arguments where required.

XQuery example
Here is an XQuery example fragment similar to the XSLT example above:

```
<math xmlns:math="clitype:System.Math">
   {math:Sqrt(9)}
</math>
```

As with the XSLT example above, the namespace declaration identifies the .NET class, in this case a system class. The XQuery expression identifies the method to be called and supplies the argument.

7.3.1 .NET: Constructors

An extension function can be used to call a .NET constructor. All constructors are called with the pseudo-function `new()`. If there is more than one constructor for a class, then the constructor that most closely matches the number of arguments supplied is selected. If no constructor is deemed to match the supplied argument/s, then a `'No constructor found'` error is returned.

Constructors that return XPath/XQuery datatypes
If the result of a .NET constructor call can be implicitly converted to XPath/XQuery datatypes, 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" />
```

 
```

7.3.2 .NET: Static Methods and Static Fields

A static method is called directly by its name and by supplying the arguments for the method. The name used in the call must exactly match a public static method in the class specified. If the method name and the number of arguments that were given in the function call matches more than one method in a class, then the types of the supplied arguments are evaluated for the best match. If a match cannot be found unambiguously, an error is reported.

Note: A field in a .NET class is considered to be a method without any argument. A property is called using the syntax `get_PropertyName()`.

Examples

An XSLT example showing a call to a method with one argument (`System.Math.Sin(arg)`):

```
<xsl:value-of select="math:Sin(30)" xmlns:math="clitype:System.Math"/>
```

An XSLT example showing a call to a field (considered a method with no argument) (`System.Double.MaxValue()`):

```
<xsl:value-of select="double:MaxValue()" xmlns:double="
clitype:System.Double"/>
```

An XSLT example showing a call to a property (syntax is `get_PropertyName()`) (`System.String()`):

```
<xsl:value-of select="string:get_Length('my string')" xmlns:string="
clitype:System.String"/>
```

An XQuery example showing a call to a method with one argument (`System.Math.Sin(arg)`):

```
<sin xmlns:math="clitype:System.Math">
   { math:Sin(30) }
</sin>
```

7.3.3 .NET: Instance Methods and Instance Fields

An instance method has a .NET object passed to it as the first argument of the method call. This .NET object typically would be created by using an extension function (for example a constructor call) or a stylesheet parameter/variable. An XSLT example of this kind would be:

```
<xsl:stylesheet version="2.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:fn="http://www.w3.org/2005/xpath-functions">
   <xsl:output method="xml" omit-xml-declaration="yes"/>
   <xsl:template match="/">
      <xsl:variable name="releasedate"
         select="date:new(2008, 4, 29)"
         xmlns:date="clitype:System.DateTime"/>
      <doc>
         <date>
            <xsl:value-of select="date:ToString(date:new(2008, 4, 29))"
               xmlns:date="clitype:System.DateTime"/>
         </date>
         <date>
            <xsl:value-of select="date:ToString($releasedate)"
               xmlns:date="clitype:System.DateTime"/>
         </date>
      </doc>
   </xsl:template>
</xsl:stylesheet>
```

In the example above, a `System.DateTime` constructor (`new(2008, 4, 29)`) is used to create a .NET object of type `System.DateTime`. This object is created twice, once as the value of the variable `releasedate`, a second time as the first and only argument of the `System.DateTime.ToString()` method. The instance method `System.DateTime.ToString()` is called twice, both times with the `System.DateTime` constructor (`new(2008, 4, 29)`) as its first and only argument. In one of these instances, the variable `releasedate` is used to get the .NET object.

Instance methods and instance fields
The difference between an instance method and an instance field is theoretical. In an instance method, a .NET object is directly passed as an argument; in an instance field, a parameter or variable is passed instead—though the parameter or variable may itself contain a .NET object. For example, in the example above, the variable `releasedate` contains a .NET object, and it is this variable that is passed as the argument of `ToString()` in the second `date` element constructor. Therefore, the `ToString()` instance in the first `date` element is an instance method while the second is considered to be an instance field. The result produced in both instances, however, is the same.

7.3.4 Datatypes: XPath/XQuery to .NET

When a .NET extension function is used within an XPath/XQuery expression, the datatypes of the function's arguments are important for determining which one of multiple .NET methods having the same name is called.

In .NET, the following rules are followed:

- If there is more than one method with the same name in a class, then the methods available for selection are reduced to those that have the same number of arguments as the function call.
- The XPath/XQuery string, number, and boolean datatypes (*see list below*) are implicitly converted to a corresponding .NET datatype. If the supplied XPath/XQuery type can be converted to more than one .NET type (for example, `xs:integer`), then that .NET type is selected which is declared for the selected method. For example, if the .NET method being called is `fx(double)` and the supplied XPath/XQuery datatype is `xs:integer`, then `xs:integer` will be converted to .NET's `double` datatype.

The table below lists the implicit conversions of XPath/XQuery string, number, and boolean types to .NET datatypes.

`xs:string`	`StringValue, string`
`xs:boolean`	`BooleanValue, bool`
`xs:integer`	`IntegerValue, decimal, long, integer, short, byte, double, float`
`xs:float`	`FloatValue, float, double`
`xs:double`	`DoubleValue, double`
`xs:decimal`	`DecimalValue, decimal, double, float`

Subtypes of the XML Schema datatypes listed above (and which are used in XPath and XQuery) will also be converted to the .NET type/s corresponding to that subtype's ancestor type.

In some cases, it might not be possible to select the correct .NET method based on the supplied information. For example, consider the following case.

- The supplied argument is an `xs:untypedAtomic` value of 10 and it is intended for the method `mymethod(float)`.
- However, there is another method in the class which takes an argument of another datatype: `mymethod(double)`.
- Since the method names are the same and the supplied type (`xs:untypedAtomic`) could be converted correctly to either `float` or `double`, it is possible that `xs:untypedAtomic` is converted to `double` instead of `float`.
- Consequently the method selected will not be the required method and might not produce the expected result. To work around this, you can create a user-defined method with a different name and use this method.

Types that are not covered in the list above (for example `xs:date`) will not be converted and will generate an error.

7.3.5 Datatypes: .NET to XPath/XQuery

When a .NET method returns a value and the datatype of the value is a string, numeric or boolean type, then it is converted to the corresponding XPath/XQuery type. For example, .NET's `decimal` datatype is converted to `xsd:decimal`.

When a .NET object or a datatype other than string, numeric or boolean is returned, you can ensure conversion to the required XPath/XQuery type by first using a .NET method (for example `System.DateTime.ToString()`) to convert the .NET object to a string. In XPath/XQuery, the string can be modified to fit the lexical representation of the required type and then converted to the required type (for example, by using the `cast as` expression).

7.4 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.

Index