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+XBRL Server (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 and XBRL 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
- HTTP Interface
- Python Interface
- Java Interface
- COM/.NET Interface
- XSLT and XQuery Engine Information
- XSLT and XQuery Extension Functions
- Altova LicenseServer

Last updated: 07/23/2013

1.1 Editions and Interfaces

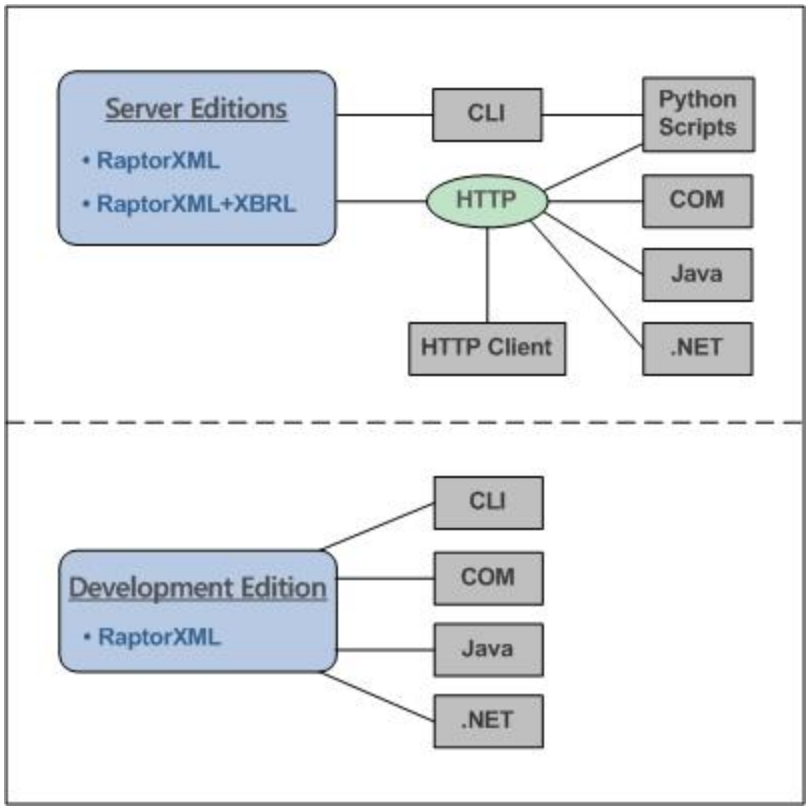
RaptorXML is available in three editions:

- *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, XBRL, DTD, XML Schema, XSLT, or XQuery files).

XML and XBRL Validation

- Validates the supplied XML or XBRL document against internal or external DTDs or XML Schemas.
- Checks well-formedness of XML, DTD, XML Schema, XSLT, and XQuery documents.
- Validates XBRL taxonomies, and XBRL documents against XBRL taxonomies.
- Execution of XBRL Formulas and Validation Assertions.
- Support for the XBRL 2.1, Dimensions 1.0, and Formula 1.0 specifications.

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

XBRL Recommendations
Website: Extensible Business Reporting Language (XBRL)

- XBRL 2.1
- Dimensions 1.0
- Formula Specification 1.0
- Formula Specification
  - Aspect Cover Filters

  - Boolean Filters
  - Concept Filters
  - Concept Relation Filters
  - Consistency Assertions
  - Custom Function Implementation
  - Dimension Filters
  - Entity Filters
  - Existence Assertions
  - Function Definition
  - General Filters
  - Generic Labels
  - Generic Messages
  - Generic References
  - Implicit Filters
  - Match Filters
  - Period Filters
  - Relative Filters
  - Segment Scenario Filters
  - Tuple Filters
  - Unit Filters
  - Validation
  - Validation Messages
  - Value Assertions
  - Value Filters
  - Variables
- Function Definitions
- Generic Link

Chapter 2

Setting Up RaptorXML

2 Setting Up RaptorXML

This section describes procedures for correctly setting up RaptorXML+XBRL Server. It describes the following:

- Installation and licensing of RaptorXML on Windows, on Linux, and on Mac OS X 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\RaptorXMLXBRLServer2013\bin\RaptorXMLXBRL.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
RaptorXML will need to be licensed with an Altova LicenseServer installed on your machine or network. (See the LicenseServer documentation for information about how to do this.) After LicenseServer has been installed, do the following. First, run the `licenseserver` command of the command line interface to register RaptorXMLXBRLServer with Altova LicenseServer. The command takes as its argument the name or IP address of the server running LicenseServer.

```
RaptorXMLXBRL licenseserver [options] Server-Or-IP-Address
```

On successfully registering RaptorXML with LicenseServer, the URL of the LicenseServer web interface will be returned. Enter the URL in a browser window to access the web interface, and then go through the licensing process as described in the LicenseServer documentation.

2.2 On Linux

RaptorXML+XBRL Server can be installed on Linux systems (CentOS 6, RedHat 6, Debian 6, and Ubuntu 12.04, or newer). This section describes how to install and license RaptorXML+XBRL Server on Linux systems:

- Installation on Linux
- Licensing RaptorXML on Linux

2.2.1 Installing on Linux

This section:

RaptorXML+XBRL Server can be installed on Linux systems (Debian, Ubuntu, CentOS).

Uninstalling old versions of RaptorXML+XBRL Server and LicenseServer
On the Linux command line interface (CLI), you can check whether RaptorXML+XBRL Server or LicenseServer is installed with the following command:

```
[Debian, Ubuntu]:   dpkg --list | grep Altova
[CentOS, RedHat]:   rpm -qa | grep server
```

If RaptorXML+XBRL Server is not installed, go ahead with the installation as documented in the next steps. If RaptorXML+XBRL Server is installed and you wish to install a newer version of RaptorXML+XBRL Server, uninstall the old version with the command:

```
[Debian, Ubuntu]:   sudo dpkg --remove raptorxmlxbrlserver
[CentOS, RedHat]:   sudo rpm -e raptorxmlxbrlserver
```

If you need to uninstall an old version of Altova LicenseServer, do this with the following command:

```
[Debian, Ubuntu]:   sudo dpkg --remove licenseserver
[CentOS, RedHat]:   sudo rpm -e licenseserver
```

Copying the Linux package
After downloading the Linux package from the Altova website, copy the package to any directory on the Linux system. Since you will need an Altova LicenseServer in order to run RaptorXML+XBRL Server, you may want to download LicenseServer from the Altova website at the same time as you download RaptorXML+XBRL Server, rather than download it at a later time.

Distribution	Installer extension
Debian	.deb
Ubuntu	.deb
CentOS	.rpm
RedHat	.rpm

Installing RaptorXML+XBRL Server

In a terminal window, switch to the directory where you have copied the Linux package. For example, if you copied it to a user directory called `MyAltova` (that is located, say, in the `/home/User` directory), then switch to this directory as follows:

```
cd /home/User/MyAltova
```

Install RaptorXML+XBRL Server with the following command:

```
[Debian]:   sudo dpkg --install raptorxmlxbrlserver-2013-debian.deb
[Ubuntu]:   sudo dpkg --install raptorxmlxbrlserver-2013-ubuntu.deb
[CentOS]:   sudo rpm -ivh raptorxmlxbrlserver-2013-1.x86_64.rpm
[RedHat]:   sudo rpm -ivh raptorxmlxbrlserver-2013-1.x86_64.rpm
```

The RaptorXML+XBRL Server package will be installed in the folder:

```
/opt/Altova/raptorxmlxbrlserver2013
```

Installing Altova LicenseServer

In order for RaptorXML+XBRL Server to run, it must be licensed via an Altova LicenseServer on your network. On Linux systems, Altova LicenseServer will need to be installed separately. Download Altova LicenseServer from the Altova website and copy the package to any directory on the Linux system. Install it just like you did RaptorXML+XBRL Server (*see previous step*).

```
[Debian]:   sudo dpkg --install licenseserver-1.1-debian.deb
[Ubuntu]:   sudo dpkg --install licenseserver-1.1-ubuntu.deb
[CentOS]:   sudo rpm -ivh licenseserver-1.1-1.x86_64.rpm
[RedHat]:   sudo rpm -ivh licenseserver-1.1-1.x86_64.rpm
```

The LicenseServer package will be installed in:

```
/opt/Altova/LicenseServer
```

For information about how to register RaptorXML+XBRL Server with Altova LicenseServer and license it, see the section, Licensing.

2.2.2 Licensing RaptorXML on Linux

This section:

Licensing procedure
To license RaptorXML+XBRL Server on Linux systems (Debian, Ubuntu, CentOS), do the following:

1. If LicenseServer is not already running as a service, start it as a service.
2. Register RaptorXML+XBRL Server with LicenseServer.
3. In the configuration page of LicenseServer, assign a license to RaptorXML+XBRL Server according to the number of cores on the RaptorXML+XBRL Server machine. How to do this is described in the Altova LicenseServer documentation.

Note: You must have both RaptorXML+XBRL Server and Altova LicenseServer installed. See the section Installation on Linux for information about installing these packages.

Note: You must have administrator (root) privileges to be able to register RaptorXML+XBRL Server with LicenseServer.

Starting LicenseServer as a service
To correctly register and license RaptorXML+XBRL Server with LicenseServer, LicenseServer must be running as a service. Start LicenseServer as a service with the following command:

```
[Debian]   sudo /etc/init.d/licenseserver start
[Ubuntu]   sudo initctl start licenseserver
[CentOS]   sudo initctl start licenseserver
[RedHat]   sudo initctl start licenseserver
```

If at any time you need to stop LicenseServer, use:

```
[Debian]   sudo /etc/init.d/licenseserver stop
[Ubuntu]   sudo initctl stop licenseserver
[CentOS]   sudo initctl stop licenseserver
[RedHat]   sudo initctl stop licenseserver
```

Registering RaptorXML+XBRL Server
Before assigning a license to RaptorXML+XBRL Server from LicenseServer, RaptorXML+XBRL Server must be registered with LicenseServer. You can register RaptorXML+XBRL Server by using the `licenseserver` command of its CLI. Note that RaptorXML+XBRL Server must be started with root rights.

```
sudo /opt/Altova/RaptorXML+XBRLServer2013/bin/RaptorXML+XBRL licenseserver
localhost
```

In the command above, `localhost` is the name of the server on which LicenseServer is installed. Notice also that the location of the RaptorXML+XBRL Server executable is:

```
/opt/Altova/RaptorXML+XBRLServer2013/bin
```

After successfully registering RaptorXML+XBRL Server, you can go to LicenseServer and assign a license to RaptorXML+XBRL Server. How to do this is described in the Altova LicenseServer documentation.

Note on cores and licenses
RaptorXML+XBRL Server licensing is based on the number of cores available on the RaptorXML+XBRL Server machine. The number of cores licensed must be greater than or equal to the number of cores available on the server, whether it's a physical or virtual machine. For example, if a server has eight cores, you must purchase at least an eight-core license.

If you are using a computer server with a large number of CPU cores but only have a low volume to process, you may also create a virtual machine that is allocated a smaller number of cores, and purchase a license for that number. Such a deployment, of course, will have less processing speed than if all available cores on the server were utilized.

2.3 On Mac OS X

RaptorXML+XBRL Server can be installed on Mac OS X systems (version 10.7 or higher). This section describes how to install and license RaptorXML+XBRL Server on Mac OS X systems:

- Installation on Mac OS X
- Licensing RaptorXML on Mac OS X

2.3.1 Installing on Mac OS X

This section:

RaptorXML+XBRL Server can be installed on Mac OS X systems (version 10.7 or higher). Since you might need to uninstall a previous version, uninstalling is described first.

Uninstalling old versions of RaptorXML+XBRL Server and LicenseServer
Before uninstalling RaptorXML+XBRL Server, stop the service with the following command:

```
sudo launchctl unload /Library/LaunchDaemons/com.altova.
RaptorXMLXBRLServer2013.plist
```

To check whether the service has been stopped, open the Activity Monitor terminal and make sure that RaptorXML+XBRL Server is not in the list.

In the Applications terminal, right-click the RaptorXML+XBRL Server icon and select **Move to Trash**. The application will be moved to Trash. You will, however, still need to remove the application from the `usr` folder. Do this with the command:

```
sudo rm -rf /usr/local/Altova/RaptorXMLXBRLServer2013/
```

If you need to uninstall an old version of Altova LicenseServer, use the same procedure outlined above for RaptorXML+XBRL Server.

```
sudo launchctl unload /Library/LaunchDaemons/com.altova.LicenserServer11.
plist
```

Downloading the Mac OS X package
After downloading the Mac OS X package from the Altova website, copy the package to any directory on the Mac OS X system. Since you will need an Altova LicenseServer in order to run RaptorXML+XBRL Server, you may want to download LicenseServer from the Altova website at the same time as you download RaptorXML+XBRL Server, rather than download it at a later time. The Mac OS X installer file has a `.pkg` file extension.

Installing RaptorXML+XBRL Server
In a terminal window, switch to the directory where you have copied the installer file, and double-click it. Go through the successive steps of the installer wizard. These are self-explanatory and include one step in which you have to agree to the license agreement

before being able to proceed.

The RaptorXML+XBRL Server package will be installed in the folder:

```
/usr/local/Altova/RaptorXMLXBRLServer2013/
```

Clicking the RaptorXML icon in the Application terminal pops up the onscreen help (this documentation).

Installing Altova LicenseServer
In order for RaptorXML+XBRL Server to run, it must be licensed via an Altova LicenseServer on your network. On Mac OS X systems, Altova LicenseServer will need to be installed separately. Download Altova LicenseServer from the Altova website and double-click the installer package to start the installation. Follow the on-screen instructions. You will need to accept the license agreement for installation to proceed.

The LicenseServer package will be installed in the folder:

```
/usr/local/Altova/LicenseServer
```

For information about how to register RaptorXML+XBRL Server with Altova LicenseServer and license it, see the section, Licensing.

2.3.2 Licensing RaptorXML on Mac OS X

This section:

Licensing procedure
To license RaptorXML+XBRL Server on Mac OS X systems, do the following:

1. If LicenseServer is not already running as a service, start LicenseServer as a service.
2. Start RaptorXMLXBRL Server as a service.
3. Register RaptorXMLXBRL Server with LicenseServer.
4. In the configuration page of LicenseServer, assign a license to RaptorXML+XBRL Server according to the number of cores on the RaptorXML+XBRL Server machine. How to do this is described in the Altova LicenseServer documentation.

Note: You must have both RaptorXML+XBRL Server and Altova LicenseServer installed and running as services. See the section Installation on Mac OS X for information about installing these packages.

Note: You must have administrator (root) privileges to be able to register RaptorXML+XBRL Server with LicenseServer.

Starting LicenseServer as a service
To correctly register and license RaptorXML+XBRL Server with LicenseServer, LicenseServer must be running as a service. Start LicenseServer as a service with the following command:

```
sudo launchctl load /Library/LaunchDaemons/com.altova.LicenserServer.plist
```

If at any time you need to stop LicenseServer, use:

```
sudo launchctl unload /Library/LaunchDaemons/com.altova.LicenserServer.plist
```

Starting RaptorXML+XBRL Server as a service
Start RaptorXML+XBRL Server as a service with the following command:

```
sudo launchctl load /Library/LaunchDaemons/com.altova.
RaptorXMLXBRLServer2013.plist
```

If at any time you need to stop RaptorXML+XBRL Server, use:

```
sudo launchctl unload /Library/LaunchDaemons/com.altova.
RaptorXMLXBRLServer2013.plist
```

Registering RaptorXML+XBRL Server
Before assigning a license to RaptorXML+XBRL Server from LicenseServer, RaptorXML+XBRL Server must be registered with LicenseServer. You can register RaptorXML+XBRL Server by using the `licenseserver` command of its CLI. Note that RaptorXML+XBRL Server must be started with root rights.

```
sudo /usr/local/Altova/RaptorXMLXBRLServer2013/bin/RaptorXMLXBRL
licenseserver localhost
```

In the command above, `localhost` is the name of the server on which LicenseServer is installed. Notice also that the location of the RaptorXML+XBRL Server executable is:

```
/usr/local/Altova/RaptorXMLXBRLServer2013/bin
```

After successfully registering RaptorXML+XBRL Server, you can go to LicenseServer and assign a license to RaptorXML+XBRL Server. How to do this is described in the Altova LicenseServer documentation.

Note on cores and licenses
RaptorXML+XBRL Server licensing is based on the number of cores available on the RaptorXML+XBRL Server machine. The number of cores licensed must be greater than or equal to the number of cores available on the server, whether it's a physical or virtual machine. For example, if a server has eight cores, you must purchase at least an eight-core license.

If you are using a computer server with a large number of CPU cores but only have a low volume to process, you may also create a virtual machine that is allocated a smaller number of cores, and purchase a license for that number. Such a deployment, of course, will have less processing speed than if all available cores on the server were utilized.

2.4 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.

2.4.1 How Catalogs Work

This section:

- Mapping public and system identifiers to local URLs
- Mapping filepaths, Web URLs, and names to local URLs

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.

2.4.2 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\RaptorXMLXBRLServer2013\etc
```

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\RaptorXMLXBRLServer2013\etc`. 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.

2.4.3 Variables for Windows System Locations

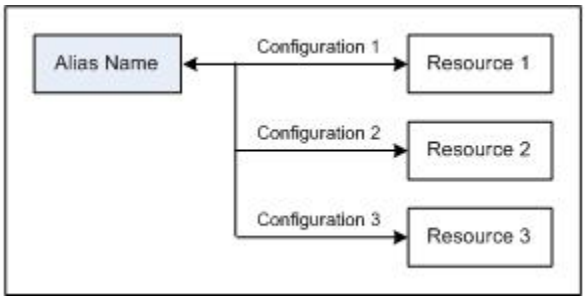
Shell environment variables can be used in catalog files to specify the path to various Windows system locations. The following variables are supported:

%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.

2.5 Global Resources

This section:

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:

```
raptorxmlxbrl 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:

```
raptorxmlxbrl 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:

```
raptorxmlxbrl 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**.

2.6 Security Issues

This section:

Some interface features of RaptorXMLXBRL Server pose security concerns. These are described below together with their solutions.

Security concerns related to the HTTP interface

The HTTP interface, by default, allows result documents to be written to any location specified by the client (that is accessible with the HTTP protocol). It is important therefore to consider this security aspect when configuring RaptorXMLXBRL Server.

If there is a concern that security might be compromised or that the interface might be misused, the server can be configured to write result documents to a dedicated output directory on the server itself. This is specified by setting the `server.unrestricted-filesystem-access` option of the server configuration file to `false`. When access is restricted in this way, the client can download result documents from the dedicated output directory with `GET` requests. Alternatively, an administrator can copy/upload result document files from the server to the target location.

Making Python scripts safe

When a Python script is specified via HTTP to RaptorXMLXBRL Server, the script will only work if it is located in the trusted directory. The script is executed from the trusted directory. Specifying a Python script from any other directory will result in an error. The trusted directory is specified in the `server.script-root-dir` setting of the server configuration file, and a trusted directory **must** be specified if you wish to use Python scripts. Make sure that all Python scripts to be used are saved in this directory.

Though all output generated by the server for HTTP job requests is written to the job output directory (which is a sub-directory of the `output-root-directory`), this limitation does not apply to Python scripts, which can write to any location. The server administrator must review the Python scripts in the trusted directory for potential vulnerability issues.

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\RaptorXMLXBRLServer2013\bin\raptorxmlxbrl.exe
```

Usage
The command line syntax is:

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

`RaptorXMLXBRL`	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.)

All 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.
valxbrl | xbrl	Validates an XBRL instance document (`.xbrl` extension).
valxbrltaxonomy | dts	Validates an XBRL taxonomy (schema) document (`.xsd` extension).
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.

XBRL validation commands

valxbrl | xbrl	Validates an XBRL instance document (`.xbrl` extension).
valxbrltaxonomy | dts	Validates an XBRL taxonomy (schema) document (`.xsd` extension).
valany	Validates any an XBRL instance or XBRL taxonomy document. Document type is detected automatically.

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 one or more XML instance documents against a DTD.

```
raptorxmlxbrl 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.

To validate multiple documents, either: (i) list the files to be validated on the CLI, with each file separated from the next by a space; or (ii) list the files to be validated in a text file (`.txt` file), with one filename per line, and supply this text file as the *InputFile* argument together with the `--listfile` option set to `true` (*see the Options list below*).

Examples

- **raptorxmlxbrl** valxml-withdtd --dtd=c:\MyDTD.dtd c:\Test.xml
- **raptorxmlxbrl** xml c:\Test.xml
- **raptorxmlxbrl** xml --verbose=true c:\Test.xml
- **raptorxmlxbrl** xml --listfile=true c:\FileList.txt

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-withdtd' command

`--dtd=FILE`

`--namespaces=true|false`

Processing options

`--listfile=true|false`

`--streaming=true|false`

Catalog options

`--catalog=FILE`

`--user-catalog=FILE`

Global resource options

`--enable-globalresources=true|false`

`--gr, --globalresourcefile=FILE`

`--gc, --globalresourceconfig=VALUE`

ZIP file options

`--recurse=true|false`

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

3.1.2 valxml-withxsd (xsi)

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

```
raptorxmlxbrl 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.

To validate multiple documents, either: (i) list the files to be validated on the CLI, with each file separated from the next by a space; or (ii) list the files to be validated in a text file (.txt file), with one filename per line, and supply this text file as the *InputFile* argument together with the --listfile option set to true (*see the Options list below*).

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

Examples

- **raptorxmlxbrl** valxml-withxsd --schemalocation-hints=false --xsd=c:\MyXSD.xsd c:\HasNoXSDRef.xml
- **raptorxmlxbrl** xsi c:\HasXSDRef.xml
- **raptorxmlxbrl** xsi --xsd-version=1.1 --listfile=true c:\FileList.txt

---

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

3.1.3 valdtd (dtd)

The **valdtd | dtd** command validates one or more DTD documents according to the XML 1.0 or XML 1.1 specification.

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

The *InputFile* argument is the DTD document to validate. To validate multiple documents, either: (i) list the files to be validated on the CLI, with each file separated from the next by a space; or (ii) list the files to be validated in a text file (`.txt` file), with one filename per line, and supply this text file as the *InputFile* argument together with the `--listfile` option set to `true` (*see the Options list below*).

Examples

- **raptorxmlxbrl** valdtd c:\Test.dtd
- **raptorxmlxbrl** dtd --verbose=true c:\Test.dtd
- **raptorxmlxbrl** dtd --listfile=true c:\FileList.txt

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.

Processing options
--listfile=true|false

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

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

ZIP file options
--recurse=true|false

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 one or more XML Schema documents (XSD documents) 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.

```
raptorxmlxbrl 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.

To validate multiple documents, either: (i) list the files to be validated on the CLI, with each file separated from the next by a space; or (ii) list the files to be validated in a text file (.txt file), with one filename per line, and supply this text file as the *InputFile* argument together with the --listfile option set to true (*see the Options list below*).

Examples

- **raptorxmlxbrl** valxsd c:\Test.xsd
- **raptorxmlxbrl** xsd --verbose=true c:\Test.xsd
- **raptorxmlxbrl** xsd --listfile=true c:\FileList.txt

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

--listfile=true|false

--script=FILE

Catalog options

--catalog=*FILE*

--user-catalog=*FILE*

Global resource options

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

ZIP file options

--recurse=true|false

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.

```
raptorxmlxbrl 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

- **raptorxmlxbrl** valany c:\Test.xml
- **raptorxmlxbrl** 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*

ZIP file options

--recurse=true|false

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 one or more XML documents for well-formedness according to the XML 1.0 or XML 1.1 specification.

```
raptorxmlxbrl wfxml [options] InputFile
```

The *InputFile* argument is the XML document to check for well-formedness. If you wish to check multiple documents, either: (i) list the files to be checked on the CLI, with each file separated from the next by a space; or (ii) list the files to be checked in a text file (`.txt` file), with one filename per line, and supply this text file as the *InputFile* argument together with the `--listfile` option set to `true` (*see the Options list below*).

Examples

- **raptorxmlxbrl** wfxml c:\Test.xml
- **raptorxmlxbrl** wfxml --verbose=true c:\Test.xml
- **raptorxmlxbrl** wfxml --listfile=true c:\FileList.txt

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.

Processing options
--listfile=true|false
--streaming=true|false
--dtd=*FILE*
--namespaces=true|false

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

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

ZIP file options
--recurse=true|false

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

3.2.2 wfdtd

The **wfdtd** command checks one or more DTD documents for well-formedness according to the XML 1.0 or XML 1.1 specification.

```
raptorxmlxbrl wfdtd [options] InputFile
```

The *InputFile* argument is the DTD document to check for well-formedness. If you wish to check multiple documents, either: (i) list the files to be checked on the CLI, with each file separated from the next by a space; or (ii) list the files to be checked in a text file (`.txt` file), with one filename per line, and supply this text file as the *InputFile* argument together with the `--listfile` option set to `true` (*see the Options list below*).

Examples

- **raptorxmlxbrl** wfdtd c:\Test.dtd
- **raptorxmlxbrl** wfdtd --verbose=true c:\Test.dtd
- **raptorxmlxbrl** wfdtd --listfile=true c:\FileList.txt

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.

Processing options
`--listfile=true|false`

Catalog options
`--catalog=FILE`
`--user-catalog=FILE`

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

ZIP file options
`--recurse=true|false`

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.

```
raptorxmlxbrl 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

- **raptorxmlxbrl** wfany c:\Test.xml
- **raptorxmlxbrl** 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*

ZIP file options
--recurse=true|false

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 XBRL Validation Commands

The XBRL validation commands can be used to validate XBRL instance documents and XBRL taxonomies according to the XBRL 2.1, Dimensions 1.0 and Formula 1.0 specifications. The available commands are listed below and described in detail in the sub-sections of this section:

valxbrl | xbrl	Validates an XBRL instance document (`.xbrl` extension).
valxbrltaxonomy | dts	Validates an XBRL taxonomy (schema) document (`.xsd` extension).
valany	Validates any one XBRL (instance or taxonomy) document. Note that this command is also used to validate XML, DTD, XSD, XSLT, or XQuery documents; the type of document submitted is detected automatically.

3.3.1 valxbrl (xbrl)

The **valxbrl | xbrl** command validates one or more XBRL instance documents according to the XBRL 2.1, Dimensions 1.0 and Formula 1.0 specifications.

```
raptorxmlxbrl valxbrl | xbrl [options] InputFile
```

The *InputFile* argument is the XBRL instance document to validate. To validate multiple documents, either: (i) list the files to be validated on the CLI, with each file separated from the next by a space; or (ii) list the files to be validated in a text file (`.txt` file), with one filename per line, and supply this text file as the *InputFile* argument together with the `--listfile` option set to `true` (*see the Options list below*).

Note: The XBRL instance document must not be nested in another XML document and must have the `xbrl` element as its root element.

```
<xbrl xmlns="http://www.xbrl.org/2003/instance"> ... </xbrl>
```

Examples

- **raptorxmlxbrl** valxbrl c:\Test.xbrl
- **raptorxmlxbrl** xbrl --formula-execution=true --formula-output=c:\FormulaOutput.xml c:\Test.xbrl
- **raptorxmlxbrl** xbrl --formula-execution --assertions-output=c:\AssertionsOutput.xml c:\Test.xbrl
- **raptorxmlxbrl** xbrl --formula-execution --formula-output=c:\FormulaOutput.xml --assertions-output=c:\AssertionsOutput.xml c:\Test.xbrl

---

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.

XBRL schema options

--dimensions=true|false
--formula=true|false
--preload-formula-schemas=true|false
--preload-xbrl-schemas=true|false
--schemalocation-hints=true|false

XBRL evaluation options

--assertions-output-format=json|xml
--assertions-output=*FILE*
--formula-execution=true|false
--formula-output=*FILE*
--formula-parameters-file=*FILE*

--formula-parameters=*JSON-ARRAY*
--validate-dts-only=true|false
--xinclude=true|false

Processing options

--listfile=true|false
--script=FILE

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*

ZIP file options

--recurse=true|false

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

3.3.2 valxbrltaxonomy (dts)

The **valxbrltaxonomy | dts** command validates one or more XBRL taxonomies (schemas) according to the XBRL 2.1, Dimensions 1.0 and Formula 1.0 specifications.

```
raptorxmlxbrl valxbrltaxonomy | dts [options] InputFile
```

The *InputFile* argument is the XBRL taxonomy to validate. To validate multiple documents, either: (i) list the files to be validated on the CLI, with each file separated from the next by a space; or (ii) list the files to be validated in a text file (`.txt` file), with one filename per line, and supply this text file as the *InputFile* argument together with the `--listfile` option set to `true` (*see the Options list below*).

Examples

- **raptorxmlxbrl** valxbrltaxonomy c:\Test.xsd
- **raptorxmlxbrl** dts --listfile c:\FileList.txt

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.

XBRL schema options

--dimensions=true|false
--formula=true|false
--preload-formula-schemas=true|false
--preload-xbrl-schemas=true|false

Processing options

--listfile=true|false
--script=FILE
--xinclude=true|false

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*

ZIP file options

--recurse=true|false

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

3.3.3 valany

The **valany** command validates an XBRL instance document or XBRL taxonomy according to the XBRL 2.1, Dimensions 1.0 and Formula 1.0 specifications. The type of document is detected automatically.

```
raptorxmlxbrl 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

- **raptorxmlxbrl** valany c:\Test.xsd
- **raptorxmlxbrl** valany --errorformat=text c:\Test.xbrl

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*

ZIP file options

--recurse=true|false

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

3.4 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.

3.4.1 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.

```
raptorxmlxbrl 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

- **raptorxmlxbrl** xslt --input=c:\Test.xml --output=c:\Output.xml c:\Test.xslt
- **raptorxmlxbrl** xslt --template-entry-point=StartTemplate --output=c:\Output.xml c:\Test.xslt
- **raptorxmlxbrl** 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

3.4.2 valxslt

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

```
raptorxmlxbrl 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

- **raptorxmlxbrl** valxslt c:\Test.xslt
- **raptorxmlxbrl** 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.

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

3.5 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.

3.5.1 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.

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

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

Examples

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

3.5.2 valxquery

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

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

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

Examples

- **raptorxmlxbrl** valxquery c:\Test.xquery
- **raptorxmlxbrl** 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

3.6 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

3.6.1 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.

```
raptorxmlxbrl 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:

```
RaptorXMLXBRL 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:

```
RaptorXMLXBRL valany --help
```

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

```
RaptorXMLXBRL help valany
```

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

3.6.2 License Commands

The `licenseserver` command registers RaptorXMLXBRLServer with Altova LicenseServer. It takes as its argument the name or IP address of the server running LicenseServer.

```
RaptorXMLXBRL licenseserver [options] Server-Or-IP-Address
```

On successfully registering RaptorXML with LicenseServer, the URL of the LicenseServer web interface will be returned. Enter the URL in a browser window to access the web interface, and then go through the licensing process as described in the LicenseServer documentation.

Example
Here's an example of the `licenseserver` command:

```
RaptorXMLXBRL licenseserver DOC.altova.com
```

The command specifies that the machine named `DOC.altova.com` is the machine running Altova LicenseServer.

Options
The following options are listed available:

`--j|json=true|false`
Prints the result of the registration attempt as a machine-parsable JSON object.

`--h|help`
Displays the command's help text.

`--version`
Displays the version number of RaptorXML. The option should be placed before the command. So: `RaptorXMLXBRL --version licenseserver`.

3.7 Localization Commands

You can create a localized version of the RaptorXML application for any language of your choice. Four localized versions (English, German, Spanish, and Japanese) are already available in the `<ProgramFilesFolder>\Altova\RaptorXMLXBRLServer2013\bin\` folder. These four language versions therefore do not need to be created.

Create a localized version in another language as follows:

1. Generate an XML file containing the resource strings. Do this with the `exportresourcestrings` command. The resource strings in the generated XML file will be one of the four supported languages: English (`en`), German (`de`), Spanish (`es`), or Japanese (`ja`), according to the argument used with the command.
2. Translate the resource strings from the language of the generated XML file into the target language. The resource strings are the contents of the `<string>` elements in the XML file. Do not translate variables in curly brackets, such as `{option}` or `{product}`.
3. Contact Altova Support to generate a localized RaptorXML DLL file from your translated XML file.
4. After you receive your localized DLL file from Altova Support, save the DLL in the `<ProgramFilesFolder>\Altova\RaptorXMLXBRLServer2013\bin\` folder. Your DLL file will have a name of the form `RaptorXMLXBRLServer_lc.dll`. The `_lc` part of the name contains the language code. For example, in `RaptorXMLXBRLServer_de.dll`, the `de` part is the language code for German (Deutsch).
5. Run the `setdeflang` command to set your localized DLL file as the RaptorXML application to use. For the argument of the `setdeflang` command, use the language code that is part of the DLL name.

Note: Altova RaptorXML+XBRL Server is delivered with support for four languages: English, German, Spanish, and Japanese. So you do not need to create a localized version of these languages. To set any of these four languages as the default language, use the CLI's `setdeflang` command.

3.7.1 exportresourcestrings

The **exportresourcestrings** command outputs an XML file containing the RaptorXML resource strings. The command takes two arguments: (i) the language of the resource strings in the output XML file, and (ii) the path and name of the output XML file. Allowed export languages (with their language codes in parentheses) are: English (en), German, (de), Spanish (es), and Japanese (ja).

```
raptorxmlxbrl exportresourcestrings LanguageCode XMLOutputFile
```

Note: On Linux systems, use an all-lowercase **raptorxmlxbrl** to call the executable.

Arguments
The exportresourcestrings command takes the following arguments:

LanguageCode	Specifies the target language of the export, that is, the language of resource strings in the exported XML file. Supported languages are: en, de, es, ja
XMLOutputFile	Specifies the location and name of the exported XML file..

Example
This command creates a file called Strings.xml at c:\ that contains all the resource strings of the RaptorXML application translated into German.

```
raptorxmlxbrl exportresourcestrings de c:\Strings.xml
```


3.7.2 setdeflang

The **setdeflang** command (short form is sdl) sets the default language of RaptorXML. It takes a mandatory LanguageCode argument.

```
raptorxmlxbrl setdeflang | sdl LangaugeCode
```

Note: On Linux systems, use an all-lowercase **raptorxmlxbrl** to call the executable.

Example
This command sets the default language of the application's messages to German.

```
raptorxmlxbrl setdeflang de
```

Supported languages
The table below lists the languages currently supported together with their language codes.

en	English
de	German
es	Spanish
ja	Japanese

3.8 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

3.8.1 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.

3.8.2 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.

3.8.3 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.

3.8.4 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.

3.8.5 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.

3.8.6 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.

3.8.7 XBRL Evaluation

[--assertions-output-format, --assertions-output-format, --formula-execution, --formula-output, --formula-parameters-file, --formula-parameters]

--assertions-output-format=json|xml
Specifies the output format of the assertion evaluation. Default is `json`.

--assertions-output=*FILE*
Writes the output of the assertion evaluation to the specified *FILE*.

--formula-execution=true|false
Enables evaluation of XBRL formulas. Default is `true`.

--formula-output=*FILE*
Writes the output of formula evaluation to the specified *FILE*.

--formula-parameters-file=*FILE*
Specifies a *FILE* containing the parameters for XBRL formula evaluation.

--formula-parameters=*JSON-ARRAY*
Specifies the parameters for XBRL formula evaluation as an array of JSON maps.

--validate-dts-only=true|false
The DTS is discovered by starting from the XBRL instance document. All referenced taxonomy schemas and linkbases are discovered and validated. The rest of the XBRL instance document is ignored. Default value is `false`.

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

3.8.8 XBRL Schemas

[--dimensions, --formula, --preload-formula-schemas, --preload-xbrl-schemas]

--dimensions=true|false
Enables XBRL Dimension 1.0 extensions. Default is `true`.

--formula=true|false
Enables XBRL Formula 1.0 extensions. Default is `true`.

--preload-formula-schemas=true|false
Preloads schemas of the XBRL Formula 1.0 specification. Default is `false`.

--preload-xbrl-schemas=true|false
Preloads schemas of the XBRL 2.1 specification. Default is `true`.

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

3.8.9 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.

3.8.10 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.

3.8.11 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 or XBRL 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+XBRL Server 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.

3.8.12 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.

3.8.13 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:*

```
raptorxmlxbrl 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.

3.8.14 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.

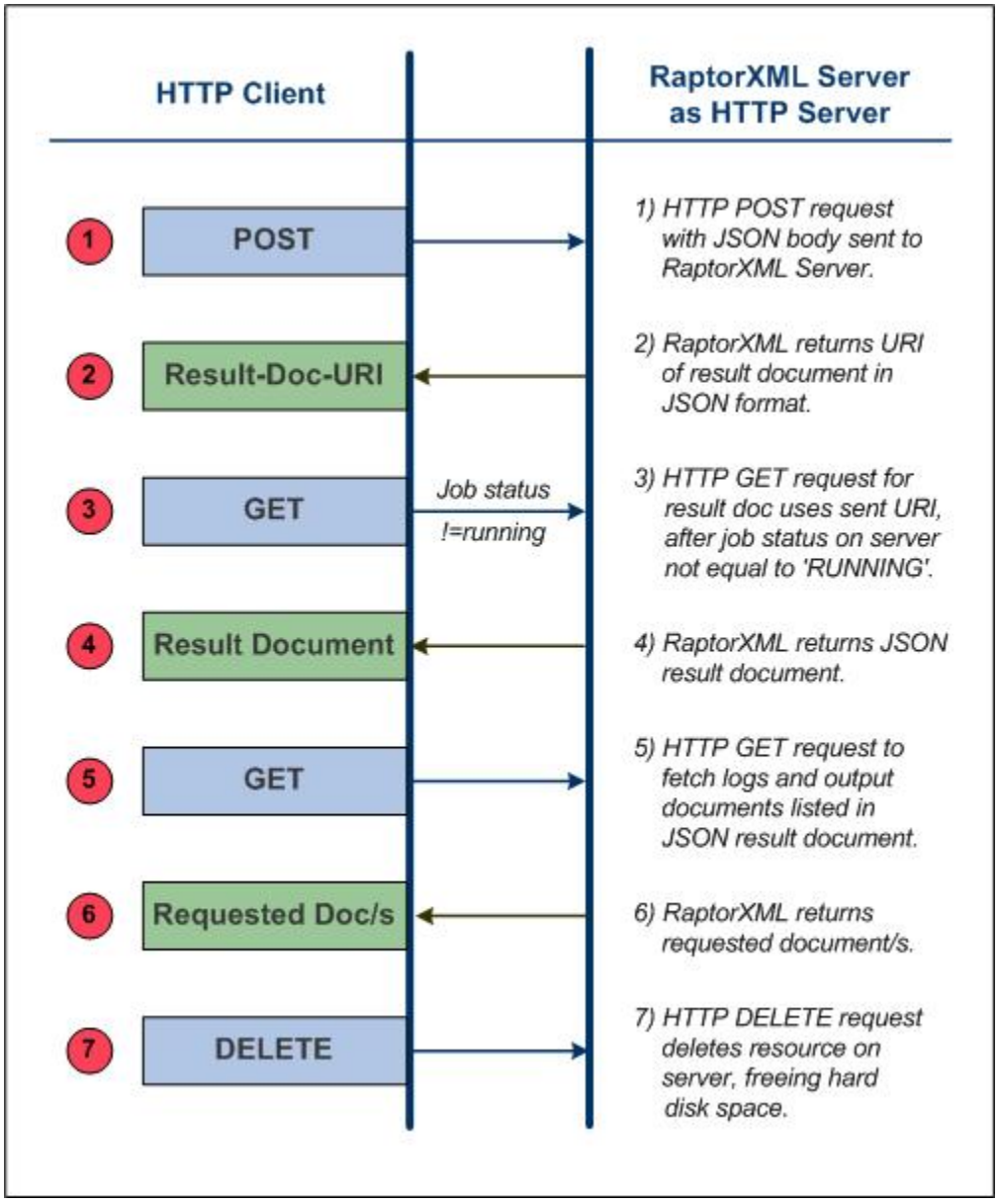
Note: Boolean option values are set to `true` if the option is specified without a value.

Chapter 4

HTTP Interface

4 HTTP Interface

RaptorXML+XBRL Server accepts validation jobs submitted via HTTP. The job description as well as the results are exchanged in JSON format. The basic workflow is as shown in the diagram below.



Security concerns related to the HTTP interface
The HTTP interface, by default, allows result documents to be written to any location specified by the client (that is accessible with the HTTP protocol). It is important therefore to consider this security aspect when configuring RaptorXMLXBRL Server.

If there is a concern that security might be compromised or that the interface might be misused, the server can be configured to write result documents to a dedicated output directory on the server itself. This is specified by setting the `server.unrestricted-filesystem-access` option of the server configuration file to `false`. When access is restricted in this way, the client can download result documents from the dedicated output directory with `GET` requests. Alternatively, an administrator can copy/upload result document files from the server to the target location.

In this section
Before sending a client request, RaptorXML+XBRL Server must be started and properly configured. How to do this is described in the section Server Setup. How to send client requests is described in the section Client Requests.

4.1 Server Setup

To correctly set up RaptorXML+XBRL Server, do the following. We assume that RaptorXML+XBRL Server has already been correctly installed and licensed.

1. RaptorXML+XBRL Server must be either started as a service or an application in order for it to be correctly accessed via HTTP. How to do this differs according to operating system and is described here: on Windows, on Linux, on Mac OS X.
2. Use the initial server configuration to test the connection to the server. (The initial server configuration is the default configuration you get on installation.) You can use a simple HTTP `GET` request like `http://localhost:8087/v1/version` to test the connection. (The request can also be typed in the address bar of a browser window.) If the service is running you must get a response to an HTTP test request such as the version request above .
3. Look at the server configuration file, `server_config.xml`. If you wish to change any settings in the file, edit the server configuration file and save the changes.
4. If you have edited the server configuration file, then restart RaptorXML+XBRL Server as a service so that the new configuration settings are applied. Test the connection again to make sure that the service is running and accessible.

Note: Server startup errors, the server configuration file used, and license errors are reported in the system log. So, refer to the system log if there are problems with the server.

4.1.1 Starting the Server

This section:

Location of the Server executable file
The RaptorXML+XBRL Server executable is installed by default in the folder:

```
<ProgramFilesFolder>\Altova\RaptorXMLXBRLServer2013\bin\RaptorXMLXBRLServer.exe
```

The executable can be used to start RaptorXML+XBRL Server as a service.

Starting as a service on Windows
The installation process will have registered RaptorXML+XBRL Server as a service on Windows. You must, however, **start** RaptorXML+XBRL Server as a service. You can do this in the following ways:

- Via the Altova ServiceController, which is available as an icon in the system tray. If the icon is not available, you can start Altova ServiceController and add its icon to the system tray by going to the **Start** menu, then selecting **All Programs | Altova | Altova LicenseServer | Altova ServiceController**.
- Via the Windows Services Management Console: **Control Panel | All Control Panel Items | Administrative Tools | Services**.
- Via the command prompt started with administrator rights. Use the following command under any directory: **`net start "Altova RaptorXML+XBRL Server"`**
- Via the RaptorXML+XBRL Server executable in a command prompt window: **`RaptorXMLXBRLServer.exe debug`**. This starts the server, with server activity information going directly to the command prompt window. The display of server activity information can be turned on and off with the `http.log-screen` setting of the server configuration file. To stop the server, press **Ctrl+Break** (or **Ctrl+Pause**). When the server is started this way—rather than as a service as described in the three previous steps—the server will stop when the command line console is closed or when the user logs off.

Starting as a service on Linux
Start RaptorXML+XBRL Server as a service with the following command:

```
[Debian]           sudo /etc/init.d/raptorxmlxbrlserver start
[Ubuntu, CentOS]   sudo initctl start raptorxmlxbrlserver
```

If at any time you need to stop RaptorXML+XBRL Server, use:

```
[Debian]           sudo /etc/init.d/raptorxmlxbrlserver stop
[Ubuntu, CentOS]   sudo initctl stop raptorxmlxbrlserver
```

Starting as a service on Mac OS X
Start RaptorXML+XBRL Server as a service with the following command:

```
sudo launchctl load /Library/LaunchDaemons/com.altova.
RaptorXMLXBRLServer2013.plist
```

If at any time you need to stop RaptorXML+XBRL Server, use:

```
sudo launchctl unload /Library/LaunchDaemons/com.altova.
RaptorXMLXBRLServer2013.plist
```

4.1.2 Testing the Connection

This section:

GET request to test the connection
After RaptorXML+XBRL Server has been started, test the connection using a `GET` request. (You can also type this request in the address bar of a browser window.)

```
http://localhost:8087/v1/version
```

Note: The interface and port number of RaptorXML+XBRL Server is specified in the server configuration file, `server_config.xml`, which is described in the next section, Server Configuration.

Server response and JSON data structure listing
If the service is running and the server is correctly configured, the request should never fail. RaptorXML+XBRL Server will return its version information as a JSON data structure (*listing below*).

```
{
  "copyright": "Copyright (c) 1998-2013 Altova GmbH. ...",
  "name": "Altova RaptorXML+XBRL Server 2013 rel. 2 sp1",
  "eula": "http://www.altova.com/server_software_license_agreement.html"
}
```

Note: If you modify the server configuration—by editing the server configuration file—you should test the connection again.

4.1.3 Configuring the Server

This section:

Server configuration file: initial settings
RaptorXML+XBRL Server is configured by means of a configuration file called `server_config.xml`, which is located by default at:

```
C:\Program Files (x86)\Altova\RaptorXMLXBRLServer2013\etc\server_config.xml
```

The initial configuration for RaptorXML+XBRL Server defines the following:

- A port number of `8087` as the server's port.
- That the server listens only for local connections (`localhost`).
- That the server writes output to `C:\ProgramData\Altova\RaptorXMLwithXBRLServer2013\Output\`.

Other default settings are shown in the listing of `server_config.xml` below.

Server configuration file: modifying the initial settings, reverting to initial settings
If you wish to change the initial settings, you must edit the server configuration file, `server_config.xml` (see listing below), save it, and then restart RaptorXML+XBRL Server as a service.

If you wish to recreate the original server configuration file (so that the server is configured with the initial settings again), run the command `createconfig`:

```
RaptorXMLXBRLServer.exe createconfig
```

On running this command, the initial settings file will be recreated and will overwrite the file `server_config.xml`. The `createconfig` command is useful if you wish to reset server configuration to the initial settings.

Server configuration file: listing and settings
The server configuration file, **`server_config.xml`**, is listed below with initial settings. Settings available in it are explained below the listing.

server_config.xml

```
<config xmlns="http://www.altova.com/schemas/altova/raptorxml/config"
  xsi:schemaLocation="http://www.altova.com/schemas/altova/raptorxml/config
  http://www.altova.com/schemas/altova/raptorxml/config.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <language>en</language>
  <server.unrestricted-filesystem-access>true</
  server.unrestricted-filesystem-access>
  <server.output-root-dir>C:\ProgramData\Altova\RaptorXMLwithXBRL
  Server2013\Output\</server.output-root-dir>
  <server.script-root-dir>C:\Program Files
  (x86)\Altova\RaptorXMLXBRLServer2013\etc\scripts\</server.script-root-dir>
  <!--<server.catalog-file>catalog.xml</server.catalog-file>-->
  <server.log-file>C:\ProgramData\Altova\RaptorXMLwithXBRL
  Server2013\Log\server.log</server.log-file>
  <http.environment>production</http.environment>
  <!--<http.socket-host>localhost</http.socket-host>-->
  <http.socket-port>8087</http.socket-port>
  <http.log-screen>true</http.log-screen>
  <http.access-file>C:\ProgramData\Altova\RaptorXMLwithXBRL
  Server2013\Log\access.log</http.access-file>
  <http.error-file>C:\ProgramData\Altova\RaptorXMLwithXBRL
  Server2013\Log\error.log</http.error-file>

</config>
```

Settings

language
Sets the language of server messages, in an optional `language` element. The default value is `en` (English). Allowed values are `en|de|es|ja` (English, German, Spanish, and Japanese, respectively). See Localization Commands for an overview of how to localize RaptorXML.

server.unrestricted-filesystem-access
When set to `true` (the default value), output files will be written directly to the location specified by the user and in Python scripts (possibly overwriting existing files of the same name). When set to `false`, files will be written to the job's directory in the output directory, and the URI of the file will be included in the result document. Setting the value to `false` provides a layer of security, since files can be written to disk only in a dedicated and known job directory on the server. Job output files can subsequently be copied by trusted means to other locations.

server.output-root-dir
Directory in which the output of all submitted jobs is saved.

server.script-root-dir
Directory in which trusted Python scripts are to be saved. The `script` option, when used via the HTTP interface, will only work when scripts from this trusted directory are used. Specifying a Python script from any other directory will result in an error. *See 'Making Python Scripts Safe'*.

server.catalog-file
URL of the XML catalog file to use. By default, the catalog file `RootCatalog.xml`, which is located in the folder `<ProgramFilesFolder>\Altova\RaptorXMLXBRLServer2013\etc`, will be

used. Use the `server.catalog-file` setting only if you wish to change the default catalog file.

`server.log-file`
Name and location of the server log file. Events on the server, like *Server started/stopped*, are logged continuously in the system's event log and displayed in a system event viewer such as Windows Event Viewer. In addition to the viewer display, log messages can also be written to the file specified with the `server.log-file` option. The server log file will contain information about all activities on the server, including server startup errors, the configuration file used, and license errors.

`http.environment`
Internal environments of raptorxml: `production | development`. The Development environment will be more geared to the needs of developers, allowing easier debugging than when the Production environment is used.

`http.socket-host`
The interface via which RaptorXML+XBRL Server is accessed. If you wish RaptorXML+XBRL Server to accept connections from remote machines, uncomment the element and set its content to: `0.0.0.0`, like this: `<http.socket-host>0.0.0.0</http.socket-host>`. This hosts the service on every addressable interface of the server machine.

`http.socket-port`
The port via which the service is accessed. The port must be fixed and known so that HTTP requests can be correctly addressed to the service.

`http.log-screen`
If RaptorXML+XBRL Server is started with the command `RaptorXMLXBRLServer.exe debug`, (see Starting the Server) and if `http.log-screen` is set to `true`, then server activity is displayed in the command line console. Otherwise server activity is not displayed. The log screen is displayed in addition to the writing of log files.

`http.access-file`
Name and location of the HTTP access file. The access file contains information about access-related activity. It contains information that is useful for resolving connection issues.

`http.error-file`
Name and location of the HTTP error file. The error file contains errors related to traffic to and from the server. If there are connection problems, this file can provide useful information towards resolving them.

The RaptorXML+XBRL Server address
The HTTP address of the server consists of the socket-host and socket-port:

```
http://{socket-host}:{socket-port}/
```


The address as set up with the initial configuration will be:
`http://localhost:8087/`

To change the address, modify the `http.socket-host` and `http.socket-port` settings in the server configuration file, `server_config.xml`. For example, say the server machine has an IP address of `100.60.300.6`, and that the following server configuration settings have been made:

```
<http.socket-host>0.0.0.0</http.socket-host>
<http.socket-port>8087</http.socket-port>
```

RaptorXML+XBRL Server can then be addressed with:
`http://100.60.300.6:8087/`

Note: After `server_config.xml` has been modified, RaptorXML+XBRL Server must be restarted for the new values to be applied.

Note: If there are problems connecting to RaptorXML+XBRL Server, information in the files named in `http.access-file` and `http.error-file` can help resolve issues.

Note: Messages submitted to RaptorXML+XBRL Server must contain path names that are valid on the server machine. Documents on the server machine can be accessed either locally or remotely (in the latter case with HTTP URIs, for example).

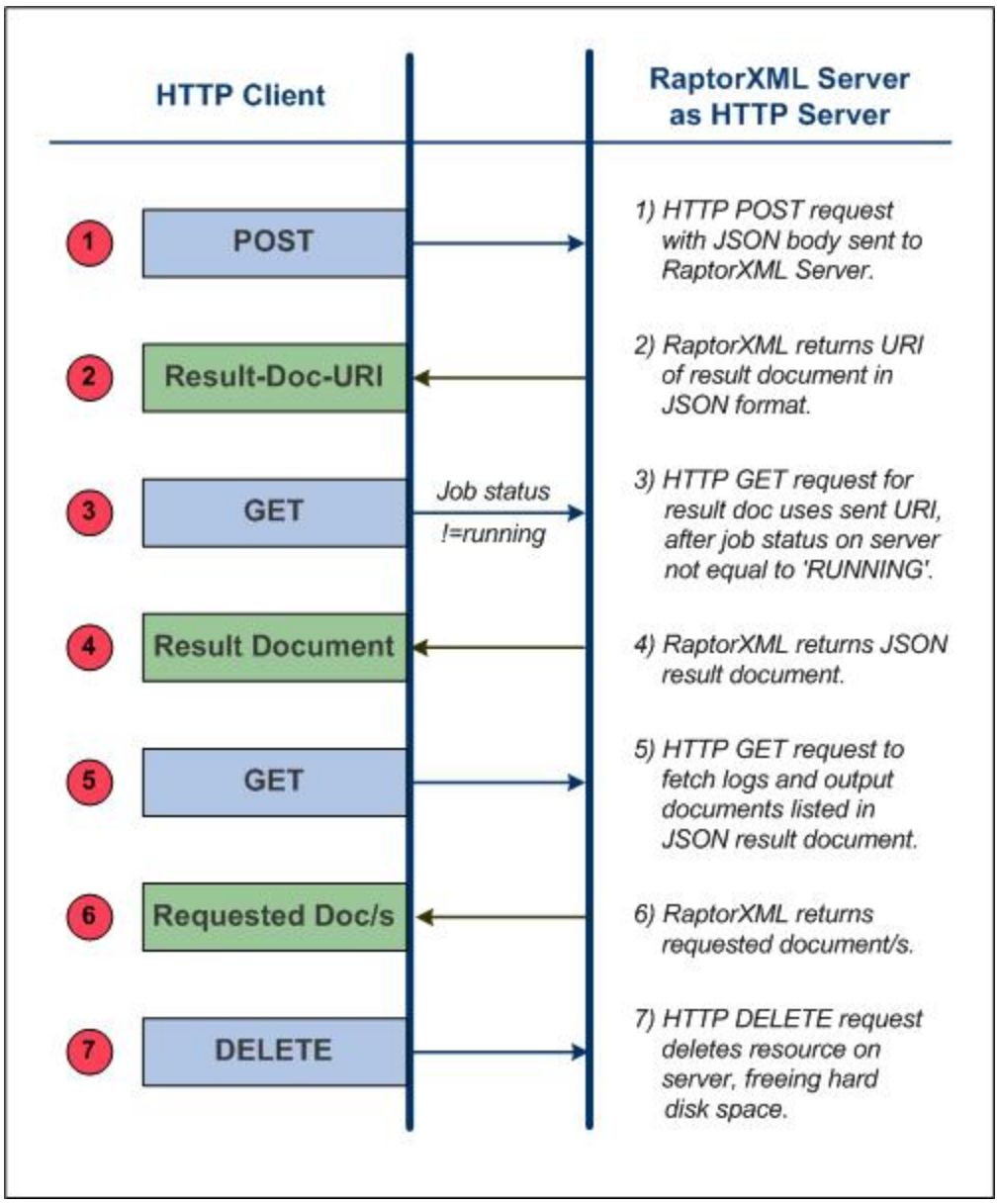
4.2 Client Requests

After RaptorXML+XBRL Server has been started as a service, its functionality can be accessed by any HTTP client which can:

- use the HTTP methods `GET`, `PUT`, `POST`, and `DELETE`
- set the `Content-Type` header field

An easy-to-use HTTP client
There are a number of web clients available for download from the Internet. An easy-to-use and reliable web client we found was Mozilla's RESTClient, which can be added as a Firefox plugin. It's easy to install, supports the HTTP methods required by RaptorXML, and provides sufficiently good JSON syntax coloring. If you have no previous experience with HTTP clients, you might want to try RESTClient. Note, however, that installation and usage of RESTClient is at your own risk.

A typical client request would consist of a series of steps as shown in the diagram below.



The important points about each step are noted below. Key terms are in bold.

1. An HTTP `POST` method is used to make a request, with the body of the request being in **JSON format**. The request could be for any functionality of RaptorXML+XBRL Server. For example, the request could be for a validation, or for an XSLT transformation. The commands, arguments, and options used in the request are the same as those used on the command line. The request is posted to: **`http://localhost:8087/v1/queue`**, assuming **`localhost:8087`** is the address of RaptorXML+XBRL Server (the initial address of the server). Such a request is termed a **RaptorXML+XBRL Server job**.

2. If the request is received and accepted for processing by RaptorXML+XBRL Server, a **result document** containing the results of the server action will be created after the job has been processed. The **URI of this result document** (the Result-Doc-URI in the diagram above), is returned to the client. Note that the URI will be returned immediately

after the job has been accepted (queued) for processing and even if processing has not been completed.

3. The client sends a request for the result document (using the result document URI) in a `GET` method to the server. If processing of the job has not yet started or has not yet been completed at the time the request is received, the server returns a status of *Running*. The `GET` request must be repeated till such time that job processing has been completed and the result document been created.

4. RaptorXML+XBRL Server returns the result document in JSON format. The result document might contain the **URIs of error or output documents** produced by RaptorXML+XBRL Server's processing of the original request. Error logs are returned, for example, if a validation returned errors. Primary output documents, such as the result of an XSLT transformation, are returned if an output-producing job is completed successfully.

5. The client sends the URIs of the output documents received in Step 4 via an HTTP `GET` method to the server. Each request is sent in a separate `GET` method.

6. RaptorXML+XBRL Server returns the requested documents in response to the `GET` requests made in Step 5.

7. The client can delete unwanted documents on the server that were generated as a result of a job request. This is done by submitting, in an HTTP `DELETE` method, the URI of the result document in question. All files on disk related to that job are deleted. This includes the result document file, any temporary files, and error and output document files. This step is useful for freeing up space on the server's hard disk.

The details of each step are described in the sub-sections of this section.

4.2.1 Initiating Jobs with POST

This section:

Sending the request
A RaptorXML+XBRL Server job is initiated with the HTTP `POST` method

HTTP Method	URI	Content-Type	Body
`POST`	`http://localhost:8087/v1/queue/`	`application/json`	JSON

Note the following points:

- The URI above has a server address that uses the settings of the initial configuration.
- The URI has a `/v1/queue/` path, which must be present in the URI. It can be considered to be an abstract folder in memory into which the job is placed.
- The correct version number `/vN` is the one that the server returns (and not necessarily the one in this documentation). The number that the server returns is the version number of the current HTTP interface. Previous version numbers indicate older versions of the HTTP interface, which are still supported for backward compatibility.
- The header must contain the field: `Content-Type: application/json`. However, if you wish to upload files within the body of the `POST` request, then the message header must have its content type set to `multipart/form-data` (i.e. `Content-Type: multipart/form-data`). See the section Uploading files with the POST request for details.
- The body of the request must be in JSON format.
- Files to be processed must be on the server. So files must either be copied to the server before a request is made, or be uploaded along with the POST request. In this case the message header must have its content type set to `multipart/form-data`. See the section Uploading files with the POST request below for details.

To check the well-formedness of an XML file, the request in JSON format would look something like this:

```
{
   "command": "wfxml", "args": [ "file:///c:/Test/Report.xml" ]
}
```

Valid commands, and their arguments and options, are as documented in the Command Line section.

JSON syntax for HTTP `POST` requests

```
{
  "command": "Command-Name",
  "options": {"opt1": "opt1-value", "opt2": "opt2-value"},
  "args"   : ["file:///c:/filename1", "file:///c:/filename2"]
}
```

- All black text is fixed and must be included. This includes all braces, double quotes, colons, commas, and square brackets. Whitespace can be normalized.
- Blue italics are placeholders and stand for command names, options and option values, and argument values. Refer to the command line section for a description of the commands.
- The `command` and `args` keys are mandatory. The `options` key is optional. Some `options` keys have default values; so, of these options, only those for which the default values need to be changed need be specified.
- All strings must be enclosed in double quotes. Boolean values and numbers must not have quotes. So: {`"error-limit": "unlimited"}` and `{"error-limit": 1}` is correct usage.
- Notice that file URIs—rather than file paths—are recommended and that they use forward slashes. Windows file paths, if used, take backslashes. Furthermore, Windows file-path backslashes must be escaped in JSON (with backslash escapes; so `"c:\\dir\\filename"`). Note that file URIs and file paths are strings and, therefore, must be in quotes.

Uploading files with the POST request

Files to be processed can be uploaded within the body of the `POST` request. In this case, the `POST` request must be made as follows.

Request header

In the request header, set `Content-Type: multipart/form-data` and specify any arbitrary string as the boundary. Here is an example header:

```
Content-Type: multipart/form-data; boundary=---MyBoundary
```

The purpose of the boundary is to set the boundaries of the different form-data parts in the request body (*see below*).

Request body: Message part

The body of the request has the following form-data parts, separated by the boundary string

specified in the request header (*see above*):

- *Mandatory form-data parts:* **msg**, which specifies the processing action requested, and **args**, which contains the files to be uploaded as the argument/s of the command specified in the `msg` form-data part. *See the listing below.*
- *Optional form-data part:* A form-data part name **additional_files**, which contains files referenced from files in the `msg` or `args` form-data parts. Additionally form-data parts named after an option of the command can also contain files to be uploaded.

Note: All uploaded files are created in a single virtual directory.

Given below is a listing of the body of a `POST` request. It has numbered callouts that are explained below. The command submitted in the listing request would have a CLI equivalent of:

```
raptorxmlxbrl xsi First.xml Second.xml --xsd=Demo.xsd
```

The request is for the validation of two XML files according to a schema. The body of the request would look something like this, assuming that `---PartBoundary` has been specified in the header as the boundary string (*see Request Header above*).

```
---PartBoundary                                                          1
Content-Disposition: form-data; name="msg"
Content-Type: application/json

{"command": "xsi", "options": {}, "args": []}                            2

---PartBoundary                                                          3
Content-Disposition: form-data; name="args"
Content-Type: multipart/mixed; boundary=---ArgsBoundary

---ArgsBoundary                                                          4
  Content-Disposition: attachment; filename="First.xml"
  Content-Type: application/octet-stream

  <?xml version="1.0" encoding="UTF-8"?>                                 5
  <test xsi:noNamespaceSchemaLocation="Demo.xsd" xmlns:xsi="
  http://www.w3.org/2001/XMLSchema-instance">42</test>

---ArgsBoundary                                                          6
  Content-Disposition: attachment; filename="Second.xml"
  Content-Type: application/octet-stream

  <?xml version="1.0" encoding="UTF-8"?>                                 7
  <test xsi:noNamespaceSchemaLocation="Demo.xsd" xmlns:xsi="
  http://www.w3.org/2001/XMLSchema-instance">35</test>

---ArgsBoundary--                                                        8

---PartBoundary                                                          9
Content-Disposition: form-data; name="additional_files"
Content-Type: multipart/mixed; boundary=---AddnlFilesBoundary

---AddnlFilesBoundary                                                    10
  Content-Disposition: attachment; filename="Demo.xsd"
  Content-Type: application/octet-stream
```

```
<?xml version="1.0" encoding="UTF-8"?>                                    11
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="
unqualified">
   <xs:element name="test" type="xs:int"/>
</xs:schema>

---AddnlFilesBoundary--                                                    12
---PartBoundary--
```

1 The name of the main form-data part boundaries are declared in the request header. The first form-data part (in this example) is **msg**. Note that the content type is `application/json`.

2 This is the standard syntax for HTTP POST requests. If `args` contains a reference to a file and if additional files are uploaded, both sets of files will be passed to the server.

3 The boundary signifies both the end of the preceding `msg` form-data part and the start of a new **args** form-data part. This part can have nested parts with a boundary name of your choice; in the listing above it is named `ArgsBoundary`.

4 The first member of the `args` array is a file attachment called `First.xml`.

5 The text of the file `First.xml`. It contains a reference to a schema called `Demo.xsd`, which will also be uploaded—in the **additional_files** form-data part.

6 The second member of the `args` array is an attachment called `Second.xml`. It is demarcated from the first member of the array by an `ArgsBoundary` string.

7 The text of the file `Second.xml`. It too contains a reference to the schema `Demo.xsd`. *See callout 10.*

8 The end of the last member of the `args` array (and therefore the `args` array itself) is signified by a `'--'` suffix after the `args` boundary name.

9 The form-data part boundary indicates the end of the `args` form-data part and the start of the **additional_files** form-data part. The latter has its own differently named boundaries.

10 The first additional files part contains the `Demo.xsd` attachment metadata.

11 The text of the file `Demo.xsd`.

12 The end of the `Demo.xsd` additional files part, and the `additional_files` form-data part.

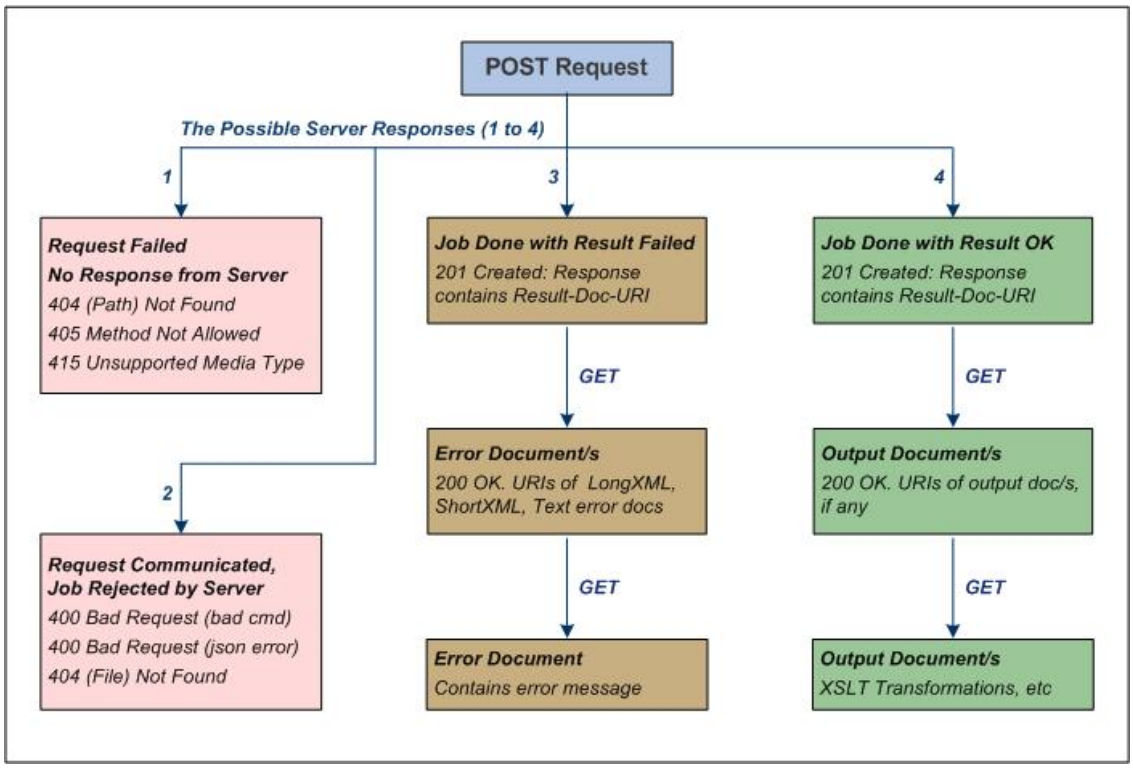
4.2.2 Server Response to POST Request

This section:

- Overview of possible server responses
- Response: Request failed, no response from server
- Response: Request communicated, but job rejected by server
- Response: Job executed (with positive or negative result)

When a `POST` request is made successfully to the server, the job is placed in the server queue. A `201 Created` message and a result document URI are returned. The job will be processed at the earliest. In the meanwhile, if the the result document is requested, a `"status": "Running"` message is returned if the job has not been completed; the client should try again at a later time. A `Dispatched` state indicates that the job is in the server queue but has not yet bee started.

The result of the job (for example, a validation request) may be negative (validation failed) or positive (validation successful). In either case a `201 Created` message is returned and a result document is generated. It is also possible that the `POST` request was not communicated to the server (*Request failed*), or the request was communicated but the job was rejected by the server (*Request communicated, but job rejected*). The various possible outcomes are shown in the diagram below.



The possible outcomes to the client's `POST` request are as follows:

Request failed, no response from server
When requests cannot be made successfully to the server, the most common errors are those listed below:

Message	Explanation
404 Not Found	The correct path is: `http://localhost:8087/v1/queue/`
405 Method Not Allowed	Specified method is invalid for this resource. Use the `POST` method.
415 Unsupported Media Type	The message header should be `Content-Type:application/json`.

Request communicated, but job rejected by server
When requests are made successfully to the server, the server could reject them for the following reasons:

Message	Explanation
400 Bad Request (*bad cmd*)	The RaptorXML command is incorrect.
400 Bad Request (*json error*)	The request body has a JSON syntax error.
404 File Not Found	Check file URI (or filepath) syntax of all files named in the command.

Job executed (with positive or negative result)
When a job (for example, a validation job) is executed, its result can be positive (*OK*) or negative (*Failed*). For example, the result of a validation job is positive (*OK*) when the document to be validated is valid, negative (*Failed*) if the document is invalid.

In both cases, the job is executed, but with different results. A `201 Created` message is returned in both cases as soon as the job is successfully placed in the queue. Also, in both cases a result document URI is returned to the HTTP client that made the request. (The result document itself might not yet have been created if processing of the job has not yet started or completed.) After the result document has been created, it can be fetched with an HTTP `GET` request. In addition to the result document, other documents may be generated also, as follows:

- *Job executed with result 'Failed':* An error log is created in three formats: text, long XML, and short XML. The URIs of these three documents are sent in the result document (which is in JSON format). The URIs can be used in an HTTP `GET` request to fetch the error documents.
- *Job executed with result 'OK':* The job is processed successfully and output documents —such as the output produced by an XSLT transformation—are created. If output files have been generated, their URIs are sent in the JSON-format result document. The URIs can then be used in an HTTP `GET` request to fetch the output documents. Note that not all jobs will have output files; for example, a validation job. Also a job can finish

with a state of '*OK*', but there might have been warnings and/or other messages that were written to error files. In this case, error file URIs are also sent in the result document (that is, in addition to output documents).

See Getting the Result Document and Getting Error/Output Documents for a description of these documents and how to access them.

4.2.3 Getting the Result Document

This section:

The Result Document URI
A result document will be created every time a job is created, no matter whether the result of a job (for example, a validation) is positive (document valid) or negative (document invalid). In both cases a `201 Created` message is returned. This message will be in JSON format and will contain a relative URI of the result document. The JSON fragment will look something like this:

```
{
  "result": "/v1/results/E6C4262D-8ADB-49CB-8693-990DF79EABEB",
  "jobid": "E6C4262D-8ADB-49CB-8693-990DF79EABEB"
}
```

The `result` object contains the relative URI of the result document. The URI is relative to the server address. For example, if the server address is `http://localhost:8087/` (the initial configuration address), then the expanded URI of the result document specified in the listing above will be:

```
http://localhost:8087/v1/results/E6C4262D-8ADB-49CB-8693-990DF79EABEB
```

Note: The correct version number `/vN` is the one that the server returns (and is not necessarily the one in this documentation). The number that the server returns is the version number of the current HTTP interface. Previous version numbers indicate older versions of the HTTP interface, which, however, are still supported for backward compatibility.

Fetching the Result Document
To get the result document submit the document's expanded URI (*see above*), in an HTTP `GET` request. The result document is returned and could be one of the generic types described below.

Note: When a job is successfully placed in the server queue, the server returns the URI of the result document. If the client requests the result before the job has been started (it is still in the queue), a `"status": "Dispatched"` message will be returned. If the job has been started but not completed (say, because it is a large job), a `"status": "Running"` message will be returned. In these two situations, the client should wait for some time before making a fresh request for the result document.

Note: The example documents below all assume restricted client access. So error documents, message documents, and output documents are all assumed to be saved

in the relevant job directory on the server. The URIs for them in the result document are therefore all relative URIs. None is a file URI (which would be the kind of URI generated in cases of unrestricted client access). For the details of these URIs, see the section Getting Error/Message/Output Documents.

Result document containing URIs of error documents

If the requested job finished with a state of *Failed*, then the job returned a negative result. For example, a validation job returned a document-invalid result. The errors encountered while executing the job are stored in error logs, created in three file formats: (i) text, (ii) long-XML (detailed error log), and (iii) short-XML (less-detailed error log). See the JSON listing below.

```
{
  "jobid": "6B4EE31B-FAC9-4834-B50A-582FABF47B58",
  "state": "Failed",
  "error":
  {
    "text": "/v1/results/6B4EE31B-FAC9-4834-B50A-582FABF47B58/error/error.txt"
  ,
    "longxml": "/v1
  /results/6B4EE31B-FAC9-4834-B50A-582FABF47B58/error/long.xml",
    "shortxml": "/v1
  /results/6B4EE31B-FAC9-4834-B50A-582FABF47B58/error/short.xml"
  },
  "jobs":
  [
    {
      "file": "file:///c:/Test/ExpReport.xml",
      "jobid": "20008201-219F-4790-BB59-C091C276FED2",
      "output":
      {
      },
      "state": "Failed",
      "error":
      {
        "text": "/v1
  /results/20008201-219F-4790-BB59-C091C276FED2/error/error.txt",
        "longxml": "/v1
  /results/20008201-219F-4790-BB59-C091C276FED2/error/long.xml",
        "shortxml": "/v1
  /results/20008201-219F-4790-BB59-C091C276FED2/error/short.xml"
      }
    }
  ]
}
```

Note the following:

- Jobs have sub-jobs.
- Errors at sub-job level propagate up to the top-level job. The state of the top-level job will be *OK* only if all of its sub-jobs have a state of *OK*.
- Each job or sub-job has its own error log.
- Error logs include warning logs. So, even though a job finishes with a state of *OK*, it might have URIs of error files.
- The URIs of the error files are relative to the server address (*see above*).

Result document containing URIs of output documents
If the requested job finished with a state of *OK*, then the job returned a positive result. For example, a validation job returned a document-valid result. If the job produced an output document—for example, the result of an XSLT transformation—then the URI of the output document is returned. See the JSON listing below.

```
{
  "jobid": "5E47A3E9-D229-42F9-83B4-CC11F8366466",
  "state": "OK",
  "error":
  {
  },
  "jobs":
  [
    {
      "file": "file:///c:/Test/SimpleExample.xml",
      "jobid": "D34B5684-C6FF-4A7A-BF35-EBB9A8A8C2C8",
      "output":
      {
        "xslt-output-file":
        [
          "/v1/results/D34B5684-C6FF-4A7A-BF35-EBB9A8A8C2C8/output/test.html"
        ]
      },
      "state": "OK",
      "output-mapping":
      {
        "/v1/results/D34B5684-C6FF-4A7A-BF35-EBB9A8A8C2C8/output/1":
        "file:///c:/temp/test.html"
      },
      "error":
      {
      }
    }
  ]
}
```

Note the following:

- The output file is created in the `output` folder of the job. You can use its relative URI to access the file.
- The URIs of the output files are relative to the server address (*see above*).
- The `output-mapping` item maps the output document in the job directory on the server to the file location specified by the client in the job request. Notice that only output documents specified by the client in the job request have a mapping; job-related files generated by the server (such as error files) have no mapping.

Result document containing no URI
If the requested job finished with a state of *OK*, then the job returned a positive result. For example, a validation job returned a document-valid result. Some jobs—such as a validation or well-formed-test—produce no output document. If a job of this type finishes with a state of *OK*, then the result document will have neither the URI of an output document nor the URI of an error log. See the JSON listing below.

```
{
  "jobid": "3FC8B90E-A2E5-427B-B9E9-27CB7BB6B405",
  "state": "OK",
```



```
  "error":
  {
  },
  "jobs":
  [
    {
      "file": "file:///c:/Test/SimpleExample.xml",
      "jobid": "532F14A9-F9F8-4FED-BCDA-16A17A848FEA",
      "output":
      {
      },
      "state": "OK",
      "error":
      {
      }
    }
  ]
}
```

Note the following:

- Both the output and error components of the sub-job in the listing above are empty.
- A job could finish with a state of *OK* but still contain warnings or other messages, which are logged in error files. In such cases, the result document will contain URIs of error files even though the job finished with a state of *OK*.

Accessing error and output documents listed in the Result Document
Error and output documents can be accessed with HTTP `GET` requests. These are described in the next section, Getting Error/Output Documents.

4.2.4 Getting Error/Message/Output Documents

A result document can contain the file URIs or relative URIs of error documents, message documents (such as logs), and/or output documents. (There are some situations in which a result document might not contain any URI.) The various kinds of URIs are described below.

To access these documents via HTTP, do the following:

1. Expand the relative URI of the file in the result document to its absolute URI
2. Use the expanded URI in an HTTP GET request to access the file

---

URIs (in the result document) of error/message/output documents
The result document contains URIs of error, message, and/or output documents. Error and message documents are job-related documents that are generated by the server; they are always saved in the job directory on the server. Output documents (such as the output of XSLT transformations) can be saved to one of the following locations:

- To any file location accessible to the server. For output files to be saved to any location, the server must be configured to allow the client unrestricted access (the default setting).
- To the job directory on the server. The server is configured to restrict client access.

If a client specifies that an output file be created, the location to which the output file is saved will be determined by the `server.unrestricted-filesystem-access` option of the server configuration file.

- If access is unrestricted, the file will be saved to the location specified by the client and the URI returned for the document will be a file URI.
- If access is restricted, the file will be saved to the job directory and its URI will be a relative URI. Additionally, there will be a mapping of this relative URI to the file URL specified by the client. (*See the listing of Result document containing URIs of output documents.*)

In summary, therefore, the following kinds of URIs will be encountered:

File URI of error/message documents
These documents are saved in the job directory on the server. File URIs will have this form:
`file:///<output-root-dir>/JOBID/message.doc`

File URI of output documents
These documents are saved at any location. File URIs will have this form:
`file:///<path-to-file>/output.doc`

HTTP URI of error/messag/output documents
These documents are saved in the job directory on the server. URIs are relative to the server address and must be expanded to the full HTTP URI. The relative will have this form:

`/vN/results/JOBID/error/error.txt`	for error documents
`/vN/results/JOBID/output/verbose.log`	for message documents
`/vN/results/JOBID/output/1`	for output documents

In the case of output documents, output mappings are given (*see example listing*). These mappings map each output document URI in the result document to the corresponding document in the client request.

Expand the relative URI
Expand the relative URI in the result document to an absolute HTTP URI by prefixing the relative URI with the server address. For example, if the server address is:

```
http://localhost:8087/
```
(the initial configuration address)

and the relative URI of an error file in the result document is:

```
/v1/results/20008201-219F-4790-BB59-C091C276FED2/error/error.txt
```

then the expanded absolute address will be

```
http://localhost:8087/v1
/results/20008201-219F-4790-BB59-C091C276FED2/error/error.txt
```

For more related information, see the sections: Configuring the Server and Getting the Result Document.

Use an HTTP `GET` request to access the file
Use the expanded URI in an HTTP `GET` request to obtain the required file. RaptorXML+XBRL Server returns the requested document.

4.2.5 Freeing Server Resources after Processing

RaptorXML+XBRL Server keeps the result document file, temporary files, and error and output document files related to a processed job on hard disk. These files can be deleted in one of two ways:

- By providing the URI of the result document with the HTTP `DELETE` method. This deletes all files related to the job indicated by the submitted result-document URI, including error and output documents.
- Manual deletion of individual files on the server by an administrator.

The structure of the URI to use with the he HTTP `DELETE` method is as shown below. Notice that the full URI consists of the server address plus the relative URI of the result document.

HTTP Method	URI
DELETE	http://localhost:8087/v1/result/D405A84A-AB96-482A-96E7-4399885FAB0F

To locate the output directory of a job on disk, construct the URI as follows:

```
[<server.output-root-dir> see server configuration file] + [jobid]
```

Note: Since a large number of error and output document files can be created, it is advisable to monitor hard disk usage and schedule deletions according to your environment and requirements.

Chapter 5

Python Interface

5 Python Interface

The Python interface of RaptorXML+XBRL Server enables data in XML documents, XML Schema documents, XBRL instance documents, and XBRL taxonomy documents to be accessed and retrieved via Python APIs for XML, XSD and XBRL. What data in the source documents to process and how to process it is specified in a Python script passed to RaptorXML+XBRL Server.

The Python APIs
The Python APIs (for XML, XSD and XBRL) provide access to the meta-information, structural information, and data contained in XML, XSD, and XBRL instance and taxonomy documents. As a result, Python scripts can be created that make use of the APIs to access and process document information. For example, a Python script can be passed to RaptorXML+XBRL Server that writes data from an XML or XBRL instance document to a database or to a CSV file.

The Python APIs are described in the sections:

- Python XML API
- Python XSD API
- Python XBRL API

Python scripts
A user-created Python script is submitted with the `--script` parameter of the following commands:

- `valxml-withxsd (xsi)`
- `valxsd (xsd)`
- `valxbrltaxonomy (dts)`
- `valxbrl (xbrl)`

These commands invoking Python scripts can be used both on the Command Line Interface (CLI) and via the HTTP Interface. The usage of Python scripts with the Python APIs of RaptorXML+XBRL Server are described in the sections Creating Python Scripts and Executing Python Scripts.

Making Python scripts safe
When a Python script is specified via HTTP to RaptorXMLXBRL Server, the script will only work if it is located in the trusted directory. The script is executed from the trusted directory. Specifying a Python script from any other directory will result in an error. The trusted directory is specified in the `server.script-root-dir` setting of the server configuration file, and a trusted directory **must** be specified if you wish to use Python scripts. Make sure that all Python scripts to be used are saved in this directory.

Though all output generated by the server for HTTP job requests is written to the job output directory (which is a sub-directory of the `output-root-directory`), this limitation does not

apply to Python scripts, which can write to any location. The server administrator must review the Python scripts in the trusted directory for potential vulnerability issues.

5.1 Creating Python Scripts

This section:

Python version
User-created Python scripts **must conform to Python 3.3.1** as this is the Python version used in the Python APIs.

Making Python scripts safe
When a Python script is specified via HTTP to RaptorXMLXBRL Server, the script will only work if it is located in the trusted directory. The script is executed from the trusted directory. Specifying a Python script from any other directory will result in an error. The trusted directory is specified in the `server.script-root-dir` setting of the server configuration file, and a trusted directory **must** be specified if you wish to use Python scripts. Make sure that all Python scripts to be used are saved in this directory.

Though all output generated by the server for HTTP job requests is written to the job output directory (which is a sub-directory of the `output-root-directory`), this limitation does not apply to Python scripts, which can write to any location. The server administrator must review the Python scripts in the trusted directory for potential vulnerability issues.

Passing a Python script to RaptorXMLXBRL Server
A Python script is passed with the `--script` parameter of the following commands:

- `valxml-withxsd (xsi)`
- `valxsd (xsd)`
- `valxbrltaxonomy (dts)`
- `valxbrl (xbrl)`

These commands can be used on the command line interface or via the HTTP interface. For examples, see the section, Executing Python Scripts.

Entry-point Python functions

The commands that allow access to the Python interface (*see list above*) are validation commands, and the Python script will be executed only if the files submitted with the command are valid. After validation has completed successfully, RaptorXML+XBRL Server will call a specific function, according to which command was executed. The called function (*see table below*), therefore, must be defined in the Python script. It must be defined with two parameters: the first is the job object, the second varies according to which command was executed (*see table*).

Command	Function called by RaptorXML+XBRL Server
valxml-withxsd (xsi)	**on_xsi_valid**(job,instance)
valxsd (xsd)	**on_xsd_valid**(job,schema)
valxbrltaxonomy (dts)	**on_dts_valid**(job,dts)
valxbrl (xbrl)	**on_xbrl_valid**(job,instance)

Simplified structure of the Python script
The broad structure of a Python script used to access the Python interface is as follows. Notice how the entry-point Python function is defined.

```
1   import os
    from altova import xml, xsd, xbrl

2   def on_xsi_valid(job,instance):
      filename = os.path.join(job.output_dir,'script_out.txt')
      job.append_output_filename(filename)
      f = open(filename,'w')
      f.write(str(type(job))+'\n')
      f.write(str(job)+'\n')
      f.write(job.output_dir+'\n')
      f.close()
      filename2 = os.path.join(job.output_dir,'script_out2.txt')
      job.append_output_filename(filename2)
      f2 = open(filename2,'w')
      print_instance(f2,instance)
      f2.close()

3   CodeBlock-1
       ...
    CodeBlock-N
```

Description of the script structure shown above:

1. Imports Python's built-in `os` module, and then the **`xml`**, **`xsd`**, **`xbrl`** modules of the **`altova`** library.
2. The **entry-point Python function** (*see below*). This could be one of: **`on_xsi_valid`**(job,instance), **`on_xsd_valid`**(job,schema), **`on_dts_valid`**(job,dts), **`on_xbrl_valid`**(job,instance).
3. Additional blocks of code, each containing function definitions or other code.

The entry-point Python function in detail
In this section, we note important points of the entry-point Python function with the help of the following entry-point function definition.

```
def on_xsi_valid(job,instance):
  filename = os.path.join(job.output_dir,'script_out.txt')
  job.append_output_filename(filename)
  f = open(filename,'w')
  f.write(str(type(job))+'\n')
  f.write(str(job)+'\n')
  f.write(job.output_dir+'\n')
  f.close()
  filename2 = os.path.join(job.output_dir,'script_out2.txt')
  job.append_output_filename(filename2)
  f2 = open(filename2,'w')
  print_instance(f2,instance)
  f2.close
```

- The line `def on_xsi_valid(job,instance):` starts the function's definition block.
- The function is called **`on_xsi_valid`**`(job,instance)` and it takes two arguments: `job` and `instance`.
- This is the function that is invoked after RaptorXML+XBRL Server has successfully executed the command `valxml-withxsd (xsi)` and found the submitted XML file/s to be valid.
- The values of the `job` and `instance` arguments are provided by RaptorXML+XBRL Server.
- The value of the `filename` variable is constructed using `job.output_dir`, the value of which, in the case of HTTP use, is specified in the server configuration file, and in the case of CLI use is the working directory.
- The `job.append_output_filename` function appends a filename to the job output.

5.2 Executing Python Scripts

Python scripts are passed to RaptorXMLXBRL Server by giving the script's URL as the value of the `--script` option. The `--script` option is supported for the following commands:

- `valxml-withxsd (xsi)`
- `valxsd (xsd)`
- `valxbrltaxonomy (dts)`
- `valxbrl (xbrl)`

These commands can be used on the command line interface or via the HTTP interface.

Examples
Here are examples of usage with the different commands:

- **raptorxmlxbrl** `xsi --script=xml.py --streaming=false c:\HasXSDRef.xml`
- **raptorxmlxbrl** `xsd --script=xsd.py c:\Test.xsd`
- **raptorxmlxbrl** `dts --script=dts.py c:\Test.xsd`
- **raptorxmlxbrl** `xbrl --script=xbrl.py c:\Test.xbrl`

Note: When using the `--script` option with the `valxml-withxsd` command, make sure to specify `--streaming=false`. Otherwise a warning saying the script was not executed is returned.

Starting the script
After the command has been successfully submitted and the file/s to be validated are found to be valid, RaptorXML+XBRL Server calls the entry-point Python function corresponding to the just-executed command and supplies it the values of the function's two arguments. If the entry-point function is defined in the script that was passed with the `--script` parameter, then execution of the script is started.

5.3 Example-Script 01: Process XML

This Python script processes data in the file **NanonullOrg.xml** (located in the `examples` folder of the RaptorXML application folder), and creates an output document called **summary.html** that contains a table summarizing the total number of shares owned by each department's employees.

The script is passed on the CLI with a command like this:

```
raptorxmlxbrl xsi --streaming=false --script=sharesummary.py NanonullOrg.xml
```

This section contains the following listings:

- The annotated Python script
- The result document produced by the script

5.3.1 Script Listing

The annotated Python script listed below processes data in the file **NanonullOrg.xml** (located in the `examples` folder of the RaptorXML application folder), and creates an output document called **summary.html**. The output document contains a table summarizing the total number of shares owned by each department's employees.

The script can be passed on the CLI with a command like this:

```
raptorxmlxbrl xsi --streaming=false --script=sharesummary.py NanonullOrg.xml
```

Note: When using the `--script` option with the `valxml-withxsd | xsi` command, make sure to specify `--streaming=false`. Otherwise a warning saying the script was not executed is returned.

Filename: sharesummary.py

```
import os
from altova import xml

def getElemTextValue(elem):
    """Returns the text content of an XML element"""

    text = ''
    for child in elem.children:
        if isinstance(child,xml.Character):
            text += child.character_code
    return text


def getChildElemsWithName(elemParent,name):
    """Returns a list of all child elements with the given name"""

    elems = []
    for child in elemParent.children:
        if isinstance(child,xml.Element) and child.local_name == name:
            elems.append(child)
    return elems


def getDepartmentName(elemDepartment):
    """Returns the name of the department specified in the <Name> element"""

    return getElemTextValue(getChildElemsWithName(elemDepartment,'Name')[0])


def getDepartmentTotalShares(elemDepartment):
    """Returns the number of shares held by each person in that department"""

    # Initialize total shares to 0
    totalShares = 0
    # Sum the shares of each <Person> within the deparment
    for elemPerson in getChildElemsWithName(elemDepartment,'Person'):
        elemShares = getChildElemsWithName(elemPerson,'Shares')
        # <Shares> element is optional, thus we need to check for its existence
        if len(elemShares):
            # Get the value of the <Shares> element, convert it to an integer
            and add it to the total sum
            totalShares += int(getElemTextValue(elemShares[0]))
```

```
    return totalShares


def calcSharesPerDepartment(instance):
    """Return a map containing the number of shares held by the persons in
    each department"""

    # Get XML root element
    elemOrgChart  = instance.document.document_element
    # Check if the root element is <OrgChart>
    if not elemOrgChart or elemOrgChart.local_name != 'OrgChart' or
    elemOrgChart.namespace_name != 'http://www.xmlspy.com/schemas/orgchart':
       # Otherwise raise error
       raise Error('This script must be used with a valid OrgChart instance!')

    mapSharesPerDepartment = {}
    # Go through each <Department> in each <Office> and set the number of
    shares held by each person in that department
     for elemOffice in getChildElemsWithName(elemOrgChart,'Office'):
       for elemDepartment in getChildElemsWithName(elemOffice,'Department'):
          mapSharesPerDepartment[getDepartmentName(elemDepartment)] =
          getDepartmentTotalShares(elemDepartment)
    return mapSharesPerDepartment


def writeSummary(mapSharesPerDepartment,filename):
    """Write a summary containing the number of shares for each department to
    the give filename"""

    # Open file for writing
    f = open(filename,'w')
    f.write('<html><title>Summary</title><body><table border="1">\n')
    f.write('<tr><th>Department</th><th>Shares</th></tr>\n')
    # Generate a table row for each department with the deparment's name and
    its total number of shares
    for name,shares in sorted(mapSharesPerDepartment.items()):
       f.write('<tr><td>%s</td><td>%d</td></tr>\n'%(name,shares))
    f.write('</table></body></html>\n')
    # Close file
    f.close()


def on_xsi_valid(job,instance):
    """This method will be automatically called by RaptorXML after successful
    validation of the XML instance"""

    # Create a 'summary.html' file in the job's ouptut directory (when run
    from the CLI this will be the current working directory)
    filename = os.path.join(job.output_dir,'summary.html')
    # Calculate the number of shares per department and write a summary to
    'summary.html'
    writeSummary(calcSharesPerDepartment(instance),filename)
    # Register the newly generated 'summary.html' output file
    job.append_output_filename(filename)
```


5.3.2 Result Document

Given below is a listing of the document `summary.html` produced by the Python script sharesummary.py.

Filename: summary.html

```
<html><title>Summary</title><body>
  <table border="1">
    <tr><th>Department</th><th>Shares</th></tr>
    <tr><td>Administration</td><td>2500</td></tr>
    <tr><td>Engineering</td><td>5500</td></tr>
    <tr><td>IT & Technical Support</td><td>1750</td></tr>
    <tr><td>Marketing</td><td>3000</td></tr>
    <tr><td>Research & Development</td><td>5500</td></tr>
  </table>
</body></html>
```

5.4 Example-Script 02: Re-format XML

The Python script in this example reformats the XML file **NanonullOrg.xml** (located in the examples folder of the RaptorXML application folder). Each element is indented with tabs and each attribute is placed on a separate line (which could make visual comparison using a differencing tool easier). The output document is called **output.xml**.

The script is passed on the CLI with a command like this:

```
raptorxmlxbrl xsi --streaming=false --script=reformat.py NanonullOrg.xml
```

This section contains the following listings:

- The annotated Python script
- The result document produced by the script

5.4.1 Script Listing

The annotated Python script listed below (`reformat.py`) reformats the XML file **`NanonullOrg.xml`** (located in the `examples` folder of the RaptorXML application folder). Each element is indented with tabs and each attribute is placed on a separate line (which could make visual comparison using a differencing tool easier). The output document is called **`output.xml`**.

The script can be passed on the CLI with a command like this:

```
raptorxmlxbrl xsi --streaming=false --script=reformat.py NanonullOrg.xml
```

Note: When using the `--script` option with the `valxml-withxsd | xsi` command, make sure to specify `--streaming=false`. Otherwise a warning saying the script was not executed is returned.

Filename: reformat.py

```
import os
from altova import xml, xsd

def writeCharacter(f,char,depth):
    """Output XML for the charater node"""

    # Ignore text nodes containing only whitespace characters
    if not char.element_content_whitespace:
        # Write the text content
        f.write("\t"*depth + char.character_code+'\n')


def writeComment(f,comment,depth):
    """Output XML for the comment node"""

    # Write the comment
    f.write("\t"*depth + '<!-- '+comment.content+'-->\n')


def writeAttribute(f,attr,depth):
    """Output XML for the attribute node (on a separate line)"""

    # Look up prefix for the namespace in the inscope namespace map
    prefix = None
    if attr.namespace_name:
        inscope = {}
        for namespace in attr.owner_element.inscope_namespaces:
            inscope[namespace.namespace_name] = namespace.prefix
        prefix = inscope[attr.namespace_name]
        if prefix:
            prefix += ':'
    if not prefix:
        prefix = ''

    # Write the attribute with its value
    f.write("\t"*depth +
    "@"+prefix+attr.local_name+"=\""+attr.normalized_value+"\"\n")


def writeNSAttribute(f,attr,depth):
    """Output XML for the namespace attribute node (on a separate line)"""

    prefix = ""
    if attr.local_name != 'xmlns':
```

```
            prefix = 'xmlns:'

        # Write the namespace attribute with its value
        f.write("\t"*depth +
        "@"+prefix+attr.local_name+"=\""+attr.normalized_value+"\"\n")


def writeChildren(f,elem,depth):
    """Output XML for all the child nodes (indented by the given depth)"""

    # Iterate over all child nodes
    for child in elem.children:
        if isinstance(child,xml.Element):
            writeElement(f,child,depth)
        elif isinstance(child,xml.Comment):
            writeComment(f,child,depth)
        elif isinstance(child,xml.Character):
            writeCharacter(f,child,depth)


def writeElement(f,elem,depth):
    """Output XML for the element node with all its child nodes (indented by
    the given depth)"""

    # Look up prefix for the namespace in the inscope namespace map
    prefix = None
    if elem.namespace_name:
        inscope = {}
        for namespace in elem.inscope_namespaces:
            inscope[namespace.namespace_name] = namespace.prefix
        prefix = inscope[elem.namespace_name]
        if prefix:
            prefix += ':'
    if not prefix:
        prefix = ''

    if len(list(elem.attributes)) + len(list(elem.namespace_attributes)) == 0:
        # Write complete start tag (without attributes)
        f.write("\t"*depth + "<"+prefix+elem.local_name+'>\n')
    else:
        # Write start tag without the closing '>'
        f.write("\t"*depth + "<"+prefix+elem.local_name+'\n')

        # Write namespace attributes on separate lines
        for attr in elem.namespace_attributes:
        writeNSAttribute(f,attr,depth+1)
        # Write attributes on separate lines
        for attr in elem.attributes:
        writeAttribute(f,attr,depth+1)
        # Close the start tag
        f.write("\t"*depth + ">\n")

    # Write all element's children
    writeChildren(f,elem,depth+1)

    # Write end tag
    f.write("\t"*depth + "</"+prefix+elem.local_name+">\n")


def writeInstance(instance,filename):
    """Ouptput XML for the given instance where each element is indented by
    tabs and each attribute is placed on a separate line"""

    # Open output file
    f = open(filename,'w')
    # Write the content of the XML instance document
```



```
    writeChildren(f,instance.document,0)
    # Close output file
    f.close()


def on_xsi_valid(job,instance):
    """This method will be automatically called by RaptorXML after successful
    validation of the XML instance"""

    # Create a 'output.xml' file in the job's ouptut directory (when run from
    the CLI this will be the current working directory)
    filename = os.path.join(job.output_dir,'output.xml')
    # Write a reformatted version of the instance XML file where each
    attribute is placed on a separate line
    writeInstance(instance,filename)
    # Register the newly generated 'output.xml' output file
    job.append_output_filename(filename)
```

5.4.2 Result Document

Given below is a listing of the document `output.xml` produced by the Python script reformat.py.

Filename: output.xml

```
<OrgChart
	@xmlns="http://www.xmlspy.com/schemas/orgchart"
	@xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	@xmlns:ipo2="http://www.altova.com/IPO"
	@xmlns:ts="http://www.xmlspy.com/schemas/textstate"
	@xsi:schemaLocation="http://www.xmlspy.com/schemas/orgchart
OrgChart.xsd"
>
	<CompanyLogo
		@href="http://www.altova.com/nanonull.gif"
	>
	</CompanyLogo>
	<Name>
		Organization Chart
	</Name>
	<Office>
		<Name>
			Nanonull, Inc.
		</Name>
		<Desc>
			<para>
				The company was established
				<Style
					@css="font-weight: bold"
				>
					in Beverly in 1995
				</Style>
				 as a privately held software company. Since 1996,
Nanonull has been actively involved in developing nanoelectronic software
technologies. It released the first version of its acclaimed
				<Style
					@css="font-style: italic"
				>
					NanoSoft Development Suite
				</Style>
				 in February 1999. Also in 1999, Nanonull increased
its capital base with investment from a consortium of private investment
firms. The company has been expanding rapidly ever since.
			</para>
			<para>
				Due to the fact that nanoelectronic software
components are new and that sales are restricted to corporate customers,
Nanonull and its product line have not received much media publicity in the
company's early years. This has however changed in recent months as trade
journals have realized the importance of this revolutionary technology.
			</para>
		</Desc>
		<Location>
			US
		</Location>
		<Address
			@xsi:type="ipo2:US-Address"
		>
			<ipo2:street
				@xmlns:ipo="http://www.altova.com/IPO"
			>
```

```
                900 Cummings Center
            </ipo2:street>
            <ipo2:city>
                Boston
            </ipo2:city>
            <ipo2:state>
                MA
            </ipo2:state>
            <ipo2:zip>
                3234
            </ipo2:zip>
        </Address>
        <Phone>
            +1 (321) 555 5155 0
        </Phone>
        <Fax>
            +1 (321) 555 5155 4
        </Fax>
        <EMail>
            office@nanonull.com
        </EMail>
        <Department>
            <Name>
                Administration
            </Name>
            <Person
                @union="fred"
            >
                <First>
                    Vernon
                </First>
                <Last>
                    Callaby
                </Last>
                <Title>
                    Office Manager
                </Title>
                <PhoneExt>
                    582
                </PhoneExt>
                <EMail>
                    v.callaby@nanonull.com
                </EMail>
                <Shares>
                    1500
                </Shares>
                <LeaveTotal>
                    25
                </LeaveTotal>
                <LeaveUsed>
                    4
                </LeaveUsed>
                <LeaveLeft>
                    21
                </LeaveLeft>
                <union>
                    3
                </union>
                <list>
                    abc def
                </list>
                <bool>
                    true
                </bool>
                <idref>
                    fred
```

```
                                </idref>
                                <idrefs>
                                        fred joe
                                </idrefs>
                                <entity>
                                        myUnparsedEntity
                                </entity>
                                <notation>
                                        Altova-Orgchart
                                </notation>
                        </Person>
                         ...
                </Department>
                 ...
        </Office>
         ...
</OrgChart>
```

5.5 Example-Script 03: XBRL Report

This Python script processes data in any XBRL taxonomy document. It creates an output document called **report.html** that contains a a list of all concept items and tuples in the taxonomy.

The script is passed on the CLI with a command like this:

```
raptorxmlxbrl dts --script=dtsreport.py AnyTaxonomy.xsd
```

This section contains the following listings:

- The annotated Python script
- The result document produced by the script

5.5.1 Script Listing

The annotated Python script listed below processes data in any XBRL taxonomy document, producing a list of all concept items and tuples in the taxonomy. The output document is called **report.html**.

The script can be passed on the CLI with a command like this:

```
raptorxmlxbrl dts --script=dtsreport.py AnyTaxonomy.xsd
```

Filename: dtsreport.py

```
import os
from altova import xml, xsd, xbrl

def getBalance(item):
    """Return the balance as string for the given item concept"""

    if item.balance == xbrl.Concept.DEBIT:
        return 'Debit'
    elif item.balance == xbrl.Concept.CREDIT:
        return 'Credit'
    else:
        return 'None'


def getPeriodType(item):
    """Return the period type as string for the given item concept"""

    if item.period_type == xbrl.Concept.INSTANT:
        return 'Instant'
    elif item.period_type == xbrl.Concept.DURATION:
        return 'Duration'
    else:
        return 'None'


def getElemTextValue(elem):
    """Return the text content of an XML element"""

    text = ''
    # Iterate through all child nodes and concatenate all character nodes
    for child in elem.children:
        if isinstance(child,xml.Character):
            text += child.character_code
    return text


def getLabel(concept):
    """Return the text of the first label connected to this concept"""

    for label in concept.label_elements:
        # Return the text value for the first connected label element
        return getElemTextValue(label)
    # If there are no labels connected to this concept return a non-breaking
    space
     return ' '


def writeItem(f,item):
    """Write some information about the item concept"""
```



```
    f.write('<h3>'+item.qname.local_name+'</h3>\n')
    f.write('<p><table border="1">\n')
    f.write('<tr><td>Name</td><td>'+item.qname.local_name+'</td></tr>')
    f.write('<tr><td>Namespace</td><td>'+item.qname.namespace_name+'</td></tr>
    ')
    f.write('<tr><td>Type</td><td>'+item.element_declaration.type_definition.n
    ame+'</td></tr>')
    f.write('<tr><td>Abstract</td><td>'+str(item.is_abstract())+'</td></tr>')
    f.write('<tr><td>Nillable</td><td>'+str(item.is_nillable())+'</td></tr>')
    f.write('<tr><td>Numeric</td><td>'+str(item.is_numeric())+'</td></tr>')
    f.write('<tr><td>Balance</td><td>'+getBalance(item)+'</td></tr>')
    f.write('<tr><td>Period Type</td><td>'+getPeriodType(item)+'</td></tr>')
    f.write('<tr><td>Label</td><td>'+getLabel(item)+'</td></tr>')
    f.write('</table></p>\n')


def writeTuple(f,tuple):
    """Write some information about the tuple concept"""

    f.write('<h3>'+tuple.qname.local_name+'</h3>\n')
    f.write('<p><table border="1">\n')
    f.write('<tr><td>Name</td><td>'+tuple.qname.local_name+'</td></tr>')
    f.write('<tr><td>Namespace</td><td>'+tuple.qname.namespace_name+'</td></tr
    >')
    f.write('<tr><td>Abstract</td><td>'+str(tuple.is_abstract())+'</td></tr>')
    f.write('<tr><td>Nillable</td><td>'+str(tuple.is_nillable())+'</td></tr>')
    f.write('<tr><td>Label</td><td>'+getLabel(tuple)+'</td></tr>')
    f.write('</table></p>\n')


def writeReport(dts,filename):
    """Write a report listing all the item and tuple concepts in the
    taxonomy"""

    # Open output file
    f = open(filename,'w')
    f.write('<html><title>Report</title><body>\n')
    # Write all item concepts
    f.write('<h1><center>Item Concepts</center></h1>\n')
    for item in dts.items:
        writeItem(f,item)
    # Write all tuple concepts
    f.write('<h1><center>Tuple Concepts</center></h1>\n')
    for tuple in dts.tuples:
        writeTuple(f,tuple)
    f.write('</body></html>\n')
    # Close output file
    f.close()


def on_dts_valid(job,dts):
    """This method will be automatically called by RaptorXMLXBRL after
    successful validation of the XBRL taxonomy"""

    # Create a 'report.html' file in the job's ouptut directory (when run from
    the CLI this will be the current working directory)
    filename = os.path.join(job.output_dir,'report.html')
    # Create report html document of the DTS taxonomy
    writeReport(dts,filename)
    # Register the newly generated 'report.html' output file
    job.append_output_filename(filename)
```

5.5.2 Result Document

Given below is a listing of the document `summary.html` produced by the Python script dtsreport.py.

Filename: report.html

```
<html><title>Report</title><body>

<h1><center>Item Concepts</center></h1>

<h3>street</h3>
<p><table border="1">
<tr><td>Name</td><td>street</td></tr><tr><td>Namespace</td><td>
http://www.example.com/test</td></tr><tr><td>Type</td><td>stringItemType</td
></tr><tr><td>Abstract</td><td>False</td></tr><tr><td>Nillable</td><td>False</
td></tr><tr><td>Numeric</td><td>False</td></tr><tr><td>Balance</td><td>None</
td></tr><tr><td>Period Type</td><td>Instant</td></tr><tr><td>Label</td><td>
 </td></tr></table></p>

<h3>city</h3>
<p><table border="1">
<tr><td>Name</td><td>city</td></tr><tr><td>Namespace</td><td>
http://www.example.com/test</td></tr><tr><td>Type</td><td>stringItemType</td
></tr><tr><td>Abstract</td><td>False</td></tr><tr><td>Nillable</td><td>False</
td></tr><tr><td>Numeric</td><td>False</td></tr><tr><td>Balance</td><td>None</
td></tr><tr><td>Period Type</td><td>Instant</td></tr><tr><td>Label</td><td>
 </td></tr></table></p>

<h3>stateOrProvince</h3>
<p><table border="1">
<tr><td>Name</td><td>stateOrProvince</td></tr><tr><td>Namespace</td><td>
http://www.example.com/test</td></tr><tr><td>Type</td><td>stringItemType</td
></tr><tr><td>Abstract</td><td>False</td></tr><tr><td>Nillable</td><td>False</
td></tr><tr><td>Numeric</td><td>False</td></tr><tr><td>Balance</td><td>None</
td></tr><tr><td>Period Type</td><td>Instant</td></tr><tr><td>Label</td><td>
 </td></tr></table></p>

<h3>country</h3>
<p><table border="1">
<tr><td>Name</td><td>country</td></tr><tr><td>Namespace</td><td>
http://www.example.com/test</td></tr><tr><td>Type</td><td>stringItemType</td
></tr><tr><td>Abstract</td><td>False</td></tr><tr><td>Nillable</td><td>False</
td></tr><tr><td>Numeric</td><td>False</td></tr><tr><td>Balance</td><td>None</
td></tr><tr><td>Period Type</td><td>Instant</td></tr><tr><td>Label</td><td>
 </td></tr></table></p>

<h3>zipOrPostalCode</h3>
<p><table border="1">
<tr><td>Name</td><td>zipOrPostalCode</td></tr><tr><td>Namespace</td><td>
http://www.example.com/test</td></tr><tr><td>Type</td><td>stringItemType</td
></tr><tr><td>Abstract</td><td>False</td></tr><tr><td>Nillable</td><td>False</
td></tr><tr><td>Numeric</td><td>False</td></tr><tr><td>Balance</td><td>None</
td></tr><tr><td>Period Type</td><td>Instant</td></tr><tr><td>Label</td><td>
 </td></tr></table></p>

<h1><center>Tuple Concepts</center></h1>
<h3>address</h3>
<p><table border="1">
<tr><td>Name</td><td>address</td></tr><tr><td>Namespace</td><td>
http://www.example.com/test</td></tr><tr><td>Abstract</td><td>False</td></tr><
tr><td>Nillable</td><td>False</td></tr><tr><td>Label</td><td> </td></tr
></table></p>
```



```
<h3>anotherAddress</h3>
<p><table border="1">
<tr><td>Name</td><td>anotherAddress</td></tr><tr><td>Namespace</td><td>
http://www.example.com/test</td></tr><tr><td>Abstract</td><td>False</td></tr><
tr><td>Nillable</td><td>False</td></tr><tr><td>Label</td><td> </td></tr
></table></p>

</body></html>
```

5.6 Python API: The Job Object

class **Job**
A `Job` class represents a validation job in RaptorXML.

The `Job` class provides the following **instance attribute** (read-only):

`Job.output_dir`
Returns a file url with the job's output directory (when in server mode) otherwise the current working directory.

The `Job` class provides the following **instance method**:

`Job.append_ouptput_filename(filename)`
Adds an additional output filename to the list of the job's output files. This list is shown in the output on the command line, and in the job's result document returned via the HTTP interface.

5.7 Python XML API

The `xml` module provides a Python interface for the XML Infoset specification. It uses the underlying C++ Infoset implementation. This Python interface enables the user to navigate the XML document tree and access information from any XML node.

Available types
The following types are available. They are described in detail in the sub-sections of this section.

class xml.Document
The `Document` class represents an XML document and exposes the properties of the Document Information Item defined in the XML Infoset specification.

class xml.Element
The `Element` class represents an XML element and exposes the properties of the Element Information Item defined in the XML Infoset specification.

class xml.Attribute
The `Attribute` class represents an XML attribute and exposes the properties of the Attribute Information Item defined in the XML Infoset specification.

class xml.NSAttribute
The `NSAttribute` class represents an XML namespace attribute and exposes the properties of the Attribute Information Item defined in the XML Infoset specification.

class xml.ProcessingInstruction
The `ProcessingInstruction` class represents an XML processing instruction and exposes the properties of the Processing Instruction Information Item defined in the XML Infoset specification.

class xml.UnexpandedEntityReference
The `UnexpandedEntityReference` class represents an unexpanded XML entity reference and exposes the properties of the Unexpanded Entity Reference Information Item defined in the XML Infoset specification.

class xml.Character
The `Character` class represents XML character data and exposes the properties of the Character Information Item defined in the XML Infoset specification.

class xml.Comment
The `Comment` class represents an XML comment and exposes the properties of the Comment Information Item defined in the XML Infoset specification.

class xml.UnparsedEntity
The `UnparsedEntity` class represents an unparsed XML entity and exposes the properties of the Unparsed Entity Information Item defined in the XML Infoset specification.

class xml.Notation
The `Notation` class represents an XML notation and exposes the properties of the Notation Information Item defined in the XML Infoset specification.

class xml.Namespace

The `Namespace` class represents an XML namespace binding and exposes the properties of the Namespace Information Item defined in the XML Infoset specification.

class xml.QName
The `QName` class represents an XML qualified name.

5.7.1 xml.Document

class **xml.Document**
A `Document` object represents an XML document information item. It provides the following instance attributes (read-only):

Document.children
An ordered list of child information items, in document order. The list contains exactly one element information item. The list also contains one processing instruction information item for each processing instruction outside the document element, and one comment information item for each comment outside the document element. Processing instructions and comments within the DTD are excluded. If there is a document type declaration, the list also contains a document type declaration information item.

Document.document_element
The element information item corresponding to the document element.

Document.notations
An unordered set of notation information items, one for each notation declared in the DTD.

Document.unparsed_entities
An unordered set of unparsed entity information items, one for each unparsed entity declared in the DTD.

Document.base_URI
The base URI of the document entity.

Document.character_encoding_scheme
The name of the character encoding scheme in which the document entity is expressed.

Document.standalone
An indication of the standalone status of the document, either `True` or `False`. This attribute is derived from the optional standalone document declaration in the XML declaration at the beginning of the document entity, and returns `None` if there is no standalone document declaration.

Document.version
A string representing the XML version of the document. This attribute is derived from the XML declaration optionally present at the beginning of the document entity, and is `None` if there is no XML declaration.

5.7.2 xml.Element

class **xml.Element**
An `Element` object represents an XML element information item. It provides the following instance attributes (read-only):

Element.namespace_name
The namespace name, if any, of the element type. If the element does not belong to a namespace, this attribute is `None`.

Element.local_name
The local part of the element-type name. This does not include any namespace prefix or following colon.

Element.prefix
The namespace prefix part of the element-type name. If the name is unprefixed, this attribute is `None`.

Element.children
An ordered list of child information items, in document order. This list contains element, processing instruction, unexpanded entity reference, character, and comment information items, one for each element, processing instruction, reference to an unprocessed external entity, data character, and comment appearing immediately within the current element. If the element is empty, this list has no members.

Element.attributes
An unordered set of attribute information items, one for each of the attributes (specified or defaulted from the DTD) of this element. Namespace declarations do not appear in this set. If the element has no attributes, this set has no members.

Element.namespace_attributes
An unordered set of attribute information items, one for each of the namespace declarations (specified or defaulted from the DTD) of this element. Declarations of the form `xmlns=""` and `xmlns:name=""`, which undeclare the default namespace and prefixes respectively, count as namespace declarations. Prefix undeclaration was added in Namespaces in XML 1.1. By definition, all namespace attributes (including those named xmlns, whose `prefix` attribute has no value) have a namespace URI of `http://www.w3.org/2000/xmlns/`. If the element has no namespace declarations, this set has no members.

Element.inscope_namespaces
An unordered set of namespace information items, one for each of the namespaces in effect for this element. This set always contains an item with the prefix xml which is implicitly bound to the namespace name `http://www.w3.org/XML/1998/namespace`. It does not contain an item with the prefix xmlns (used for declaring namespaces), since an application can never encounter an element or attribute with that prefix. The set will include namespace items corresponding to all of the members of `namespace_attributes`, except for any representing declarations of the form `xmlns=""` or `xmlns:name=""`, which do not declare a namespace but rather undeclare the default namespace and prefixes. When resolving the prefixes of qualified names this attribute should be used in preference to the `namespace_attributes` attribute

Element.base_URI
The base URI of the element.

Element.parent
The document or element information item which contains this information item in its

`children` attribute.

5.7.3 xml.Attribute

class **xml.Attribute**
An `Attribute` object represents an XML attribute information item. It represents only normal XML attributes, not special namespace binding XML attributes. It provides the following instance attributes (read-only):

Attribute.namespace_name
The namespace name, if any, of the attribute. Otherwise, this attribute is `None`.

Attribute.local_name
The local part of the attribute name. This does not include any namespace prefix or following colon.

Attribute.prefix
The namespace prefix part of the attribute name. If the name is unprefixed, this attribute is `None`.

Attribute.normalized_value
The normalized attribute value.

Attribute.specified
A flag indicating whether this attribute was actually specified in the start-tag of its element, or was defaulted from the DTD.

Attribute.owner_element
The element information item which contains this information item in its `attributes` attribute.

5.7.4 xml.NSAttribute

class **xml.NSAttribute**
An `NSAttribute` object represents only special namespace binding XML attribute information item. It provides the following instance attributes (read-only):

NSAttribute.namespace_name
The namespace name which is always `http://www.w3.org/2000/xmlns/`.

NSAttribute.local_name
The local part of the attribute name. This does not include any namespace prefix or following colon.

NSAttribute.prefix
The namespace prefix part of the attribute name. If the name is unprefixed, this attribute is `None`.

NSAttribute.normalized_value
The normalized attribute value.

NSAttribute.specified
A flag indicating whether this attribute was actually specified in the start-tag of its element, or was defaulted from the DTD.

NSAttribute.owner_element
The element information item which contains this information item in its `attributes` attribute.

5.7.5 xml.ProcessingInstruction

class **xml.ProcessingInstruction**
A `ProcessingInstruction` object represents an XML processing instruction information item. It provides the following instance attributes (read-only):

ProcessingInstruction.target
A string representing the target part of the processing instruction.

ProcessingInstruction.content
A string representing the content of the processing instruction, excluding the target and any white space immediately following it. If there is no such content, the value of this attribute will be an empty string.

ProcessingInstruction.parent
The document, element, or document type declaration information item which contains this information item in its `children` attribute.

5.7.6 xml.UnexpandedEntityReference

class **xml.UnexpandedEntityReference**
An `UnexpandedEntityReference` object represents an unexpanded XML entity reference information item. It provides the following instance attributes (read-only):

UnexpandedEntityReference.name
The name of the entity referenced.

UnexpandedEntityReference.parent
The element information item which contains this information item in its `children` attribute.

5.7.7 xml.Character

class **xml.Character**
A `Character` object represents XML character information items. It provides the following instance attributes (read-only):

Character.character_code
A string representing all merged adjecent character information item's code points.

Character.element_content_whitespace
A boolean indicating whether the character is white space appearing within element content.

Character.parent
The element information item which contains this information item in its `children` attribute.

5.7.8 xml.Comment

class **xml.Comment**
A `Comment` object represents a XML comment information item. It provides the following instance attributes (read-only):

Comment.content
A string representing the content of the comment.

Comment.parent
The document or element information item which contains this information item in its `children` attribute.

5.7.9 xml.UnparsedEntity

class **xml.UnparsedEntity**
An `UnparsedEntity` object represents an unparsed XML entity information item. It provides the following instance attributes (read-only):

UnparsedEntity.name
The name of the entity.

UnparsedEntity.system_identifier
The system identifier of the entity, as it appears in the declaration of the entity.

UnparsedEntity.public_identifier
The public identifier of the entity. If the entity has no public identifier, this attribute is `None`.

UnparsedEntity.declaration_base_URI
The base URI relative to which the system identifier should be resolved.

UnparsedEntity.notation_name
The notation name associated with the entity

UnparsedEntity.notation
The notation information item named by the notation name. If there is no declaration for a notation with that name, this attribute is `None`.

5.7.10 xml.Notation

class **xml.Notation**
A `Notation` object represents an XML notation information item. It provides the following instance attributes (read-only):

Notation.name
The name of the notation.

Notation.system_identifier
The system identifier of the notation, as it appears in the declaration of the notation. If no system identifier was specified, this attribute is `None`.

Notation.public_identifier
The public identifier of the notation. If the notation has no public identifier, this attribute is `None`.

Notation.declaration_base_URI
The base URI relative to which the system identifier should be resolved.

5.7.11 xml.Namespace

class **xml.Namespace(prefix, namespace_name)**
A `Namespace` object represents an XML namespace binding and constructs an object of class `Namespace`. All arguments are required. The `Namespace` class provides the following instance attributes:

Namespace.prefix
The prefix whose binding this item describes. Syntactically, this is the part of the attribute name following the `xmlns:` prefix. If the attribute name is simply `xmlns`, so that the declaration is of the default namespace, this attribute is `None`.

Namespace.namespace_name
The namespace name to which the prefix is bound.

5.7.12 xml.QName

class **xml.QName(local_name, namespace_name)**
A QName object represents a XML qualified name and constructs an object of class QName. All arguments are required. The QName class provides the following instance attributes:

QName.local_name
The local name part of the qualified name.

QName.namespace_name
The namespace name part of the qualified name.

5.8 Python XSD API

The **xsd** module provides a Python interface for the C++ implementation of the XML Schema data model layer. This Python interface enables the user to navigate and access the XML Schema document and the Post Schema Validation Infoset (PSVI).

Available types
The following types are available. They are described in detail in the sub-sections of this section.

class xsd.Annotation
The `Annotation` class represents represents human- and machine-targeted annotations of schema components.

class xsd.Any
An `Any` class provides for validation of attribute and element information items dependent on their namespace names and optionally on their local names.

class xsd.AnyAttribute
An `AnyAttribute` class provides for validation of attribute information items dependent on their namespace names and optionally on their local names.

class xsd.Assertion
The `Assertion` class constrains the existence and values of related elements and attributes.

class xsd.AttributeDeclaration
An `AttributeDeclaration` class provides for: (i) local validation of attribute information item values using a simple type definition, and (ii) specifying default or fixed values for attribute information items.

class xsd.AttributeGroupDefinition
An `AttributeGroupDefinition` class does not participate in validation as such, but constructs one or more complex type definitions in whole or part. Attribute groups are identified by their `name` and `target namespace`. They must be unique within an XSD schema.

class xsd.AttributePSVI
The `AttributePSVI` class contains PSVI information about an attribute.

class xsd.AttributeUse
The `AttributeUse` class represents represents human- and machine-targeted annotations of schema components.

class xsd.Block
The `Block` class is part of the definition of an element declaration in the schema.

class xsd.ComplexTypeDefinition
A `ComplexTypeDefinition` class defines the properties of a complex type through its instance attributes.

class xsd.ContentType
A `ContentType` class specifies the element's content type.

class xsd.Defined

The `Defined` class represents a keyword member of the set of values allowed for the `disallowed_names` attribute of `NamespaceConstraint`.

class xsd.DerivationMethod
A `DerivationMethod` class provides information about the derivation method.

class xsd.ENTITY
The `ENTITY` class represents the `ENTITY` attribute type of XML.

class xsd.ElementDeclaration
The `ElementDeclaration` class provides for: (i) Local validation of element information item values using a type definition; (ii) Specifying default or fixed values for element information items; (iii) Establishing uniquenesses and reference constraint relationships among the values of related elements and attributes; (iv) Controlling the substitutability of elements through the mechanism of element substitution groups.

class xsd.ElementPSVI
If the schema-validity of an element information item has been assessed, then the PSVI properties are returned in instance attributes of the `ElementPSVI` class.

class xsd.Final
A complex type with an empty specification for `Final` can be used as a base type definition for other types derived by either of extension or restriction; the explicit values `extension` and `restriction` prevent further derivations by extension and restriction respectively. If all values are specified, then the complex type is said to be final, because no further derivations are possible.

class xsd.ID
The `ID` class represents the `ID` attribute type of XML.

class xsd.IDREF
The `IDREF` class represents represents a sequence of `ID` attribute types of XML.

class xsd.ID_IDREF_binding
The `ID_IDREF_binding` class represents a binding between `ID` and `IDREF`.

class xsd.ID_IDREF_table
The `ID_IDREF_table` class represents a set of `ID-IDREF mappings`.

class xsd.IdentityConstraintDefinition
The `IdentityConstraintDefinition` class provides for uniqueness and reference constraints with respect to the contents of multiple elements and attributes.

class xsd.Instance
The `Instance` class represents the instance document.

class xsd.ModelGroup
The `ModelGroup` class specifies a sequential (`sequence`), disjunctive (`choice`) or conjunctive (`all`) interpretation of it `particles` attribute.

class xsd.ModelGroupDefinition
A `ModelGroupDefinition` class is identified by its `name` and `target namespace`. Model group identities must be unique within an XSD schema. Model group definitions do not participate in validation, but the `term` of a `Particle` may correspond in whole or in part to a `ModelGroup` from a `ModelGroupDefinition`. The `model_group` instance attribute is the `ModelGroup` for which `ModelGroupDefinition` provides a name.

class xsd.NCName
The `NCName` class represents a non-colonized name.

class xsd.NMTOKEN
The `NMTOKEN` class represents the `NMTOKEN` attribute type from XML.

class xsd.NOTATION
The `NOTATION` class represents the `NOTATION` attribute type from XML.

class xsd.Name
The `Name` class represents an XML name.

class xsd.NamespaceBinding
The `NamespaceBinding` class provides the binding of a `namespace` to a `prefix`.

class xsd.NamespaceConstraint
The `NamespaceConstraint` class provides for validation of attribute and element items that are selected according to the specified constraint.

class xsd.NotationDeclaration
A `NotationDeclaration` class specifies a valid element or attribute value. Notation declarations do not participate in validation as such. They are referenced in the course of validating strings as members of the `NOTATION` simple type. An element or attribute information item with its governing type definition or its validating type derived from the `NOTATION` simple type is valid only if its value was among the enumerations of such simple type. As a consequence such a value is required to be the `name` of a notation declaration.

class xsd.OpenContent
An `OpenContent` property record. Optional if `variety` is element-only or mixed, otherwise must be absent.

class xsd.PSVI
The `PSVI` class provides element and attribute schema-validity assessment.

class xsd.Particle
A `Particle` class contains the components which it either directly contains or indirectly contains. It directly contains the component which is the value of its `term` attribute. It indirectly contains the particles, groups, wildcards, and element declarations which are contained by the value of its `term` property.

class xsd.QName
The `QName` class represents an XML qualified name.

class xsd.Schema
The `schema` class contains a collection of schema components, e.g. type definitions and element declarations, which have a common target namespace.

class xsd.Scope
The `Scope` class represents a `Scope` property record. Required.

class xsd.Sibling
The `Sibling` class represents a keyword member of the set of values allowed for the `disallowed_names` attribute of `NamespaceConstraint`.

class xsd.SimpleTypeDefinition
The `SimpleTypeDefinition` class represents simple types identified by their `name` and

`target namespace` attributes.

class xsd.TypeAlternative
The `TypeAlternative` class is used by an `ElementDeclaration` to specify a condition (`test`) under which a particular type (`type_definition`) is used as the governing type definition for element information items governed by that `ElementDeclaration`. Each `ElementDeclaration`may have multiple Type Alternatives in its `TypeTable`.

class xsd.TypeTable
The type definition against which an element information item is validated (its governing type definition) can be different from the declared type definition}. The `TypeTable` property of an `ElementDeclaration`, which governs conditional type assignment, and the `xsi:type` attribute of an element information item can cause the governing type definition and the declared type definition to be different.

class xsd.Unbounded
The `Unbounded` class is a string value. It represents the upper value of the `maxOccurs` property.

class xsd.ValueConstraint
The `ValueConstraint` class represents a property of the `AttributeUse` class.

class xsd.XPathExpression
To check an assertion, an instance of the XPath 2.0 data model is constructed, in which the element information item being assessed is the (parentless) root node, and elements and attributes are assigned types and values according to XPath 2.0 data model construction rules. When evaluated against this data model instance, `test` evaluates to either `true` or `false`.

Special Built-in Datatype Objects
`anyAtomicType`
`anySimpleType`
`anyURI`

String Datatype Objects
`language`
`normalizedString`
`string`
`token`

Boolean Datatype Object
`boolean`

Number Datatype Objects
`byte`
`decimal`
`double`
`float`
`int`
`integer`
`long`
`negativeInteger`
`nonNegativeInteger`
`nonPositiveInteger`
`positiveInteger`
`short`
`unsignedByte`
`unsignedInt`
`unsignedLong`

unsignedShort

Duration Datatype Objects

dayTimeDuration
duration
yearMonthDuration

Date and Time Datatype Objects

date
dateTime
dateTimeStamp
gDay
gMonth
gYear
gYearMonth
time

Binary Datatype Objects

base64Binary
hexBinary

Facet Objects

assertionsFacet
enumerationFacet
fractionDigitsFacet
lengthFacet
maxExclusiveFacet
maxInclusiveFacet
maxLengthFacet
minExclusiveFacet
minInclusiveFacet
minLengthFacet
pattern
totalDigitsFacet

5.8.1 xsd.Annotation

class **xsd.Annotation**
An `Annotation` class represents human- and machine-targeted annotations of schema components. *Go to description.*

The `Annotation` class provides the following **instance attributes** (read-only)

Annotation.application_information
A sequence of Element information items intended for automatic processing

Annotation.user_information
A sequence of Element information items intended for human consumption.

5.8.2 xsd.Any

class **xsd.Any**
An `Any` class provides for validation of attribute and element information items dependent on their namespace names and optionally on their local names. *Go to description*.

The `Any` class provides the following **constants**:

Any.SKIP
No constraints at all: the item must simply be well-formed XML.

Any.STRICT
There must be a top-level declaration for the item available, or the item must have an `xsi:type`, and the item must be valid as appropriate.

Any.LAX
If the item has a uniquely determined declaration available, it must be valid with respect to that declaration. This means: validate if possible, otherwise no need to validate.

The `Any` class provides the following **instance attributes** (read-only):

Any.annotations
A sequence of `Annotation` components.

Any.namespace_constraint
A Namespace Constraint property record. Required.

Any.process_contents
Controls the impact on assessment of the information items allowed by wildcards. Takes one of `SKIP`, `STRICT`, `LAX`. Required.

5.8.3 xsd.AnyAttribute

class **xsd.AnyAttribute**
An `AnyAttribute` class provides for validation of attribute information items dependent on their namespace names and optionally on their local names. *Go to description*.

The `AnyAttribute` class provides the following **constants**:

Any.SKIP
No constraints at all: the item must simply be correct XML.

Any.STRICT
There must be a top-level declaration for the item available, or the item must have an `xsi:type`, and the item must be valid as appropriate.

Any.LAX
If the item has a uniquely determined declaration available, it must be valid with respect to that declaration. This means: validate if possible, otherwise no need to validate.

The `AnyAttribute` class provides the following **instance attributes** (read-only):

AnyAttribute.annotations
A sequence of `Annotation` components.

AnyAttribute.namespace_constraint
A Namespace Constraint property record. Required.

AnyAttribute.process_contents
Controls the impact on assessment of the information items allowed by wildcards. Takes one of `SKIP`, `STRICT`, `LAX`. Required.

5.8.4 xsd.Assertion

class **xsd.Assertion**
An `Assertion` class constrains the existence and values of related elements and attributes. *Go to description*.

The `Assertion` class provides the following **instance attributes** (read-only):

Assertion.annotations
A sequence of `Annotation` components.

Assertion.test
An XPath Expression property record. Required.

5.8.5 xsd.AttributeDeclaration

class **xsd.AttributeDeclaration**
An `AttributeDeclaration` class provides for: (i) local validation of attribute information item values using a simple type definition, and (ii) specifying default or fixed values for attribute information items. *Go to description.*

The `AttributeDeclaration` class provides the following **instance attributes** (read-only):

AttributeDeclaration.annotations
A sequence of `Annotation` components.

AttributeDeclaration.name
An `xs:NCName` value. Required.

AttributeDeclaration.target_namespace
An `xs:anyURI` value. Optional.

AttributeDeclaration.type_definition
A Simple Type Definition component. Required.

AttributeDeclaration.scope
A `Scope` property record. Required.

AttributeDeclaration.value_constraint
A `Value Constraint` property record. Optional.

AttributeDeclaration.inheritable
An `xs:boolean` value. Required.

5.8.6 xsd.AttributeGroupDefinition

class **xsd.AttributeGroupDefinition**
An `AttributeGroupDefinition` class does not participate in validation as such, but constructs one or more complex type definitions in whole or part. Attribute groups are identified by their `name` and `target namespace`. They must be unique within an XSD schema. *Go to description*.

The `AttributeGroupDefinition` class provides the following **instance attributes** (read-only):

`AttributeGroupDefinition.annotations`
A sequence of Annotation components.

`AttributeGroupDefinition.name`
An `xs:NCName` value. Required.

`AttributeGroupDefinition.target_namespace`
An `xs:anyURI` value. Optional.

`AttributeGroupDefinition.attribute_uses`
A set of `Attribute Use` components.

`AttributeGroupDefinition.attribute_wildcard`
A Wildcard component. Optional.

5.8.7 xsd.AttributePSVI

class **xsd.AttributePSVI**
The `AttributePSVI` class contains PSVI information about an attribute. *Go to description*.

The `AttributePSVI` class provides the following **instance attributes** (read-only):

AttributePSVI.validity
The appropriate case among the following: if strictly assessed and locally valid, then `valid`; if strictly assessed and locally invalid, then `invalid`; otherwise `notKnown`.

AttributePSVI.validation_attempted
The appropriate case among the following: if strictly assessed, then `full`; otherwise `none`.

AttributePSVI.attribute_declaration
An item isomorphic to the governing type definition component.

AttributePSVI.schema_normalized_value
If the attribute's normalized value is valid with respect to the governing type definition, then the normalized value as validated, otherwise absent.

AttributePSVI.schema_actual_value
If the schema normalized value is not absent, then the corresponding actual value, otherwise absent.

AttributePSVI.type_definition
An item isomorphic to the governing type definition component.

AttributePSVI.type_definition_type
`simple`.

AttributePSVI.type_definition_namespace
The target namespace of the type definition.

AttributePSVI.type_definition_anonymous
`true` if the `name` of the type definition is absent, otherwise `false`.

AttributePSVI.type_definition_name
The `name` of the type definition, if the `name` is not absent. If the type definition's `name` property is absent, then schema processors may, but need not, provide a value which uniquely identifies this type definition among those with the same target namespace.

AttributePSVI.member_type_definition
An item isomorphic to the validating type of the schema actual value

AttributePSVI.member_type_definition_namespace
The target namespace of the validating type.

AttributePSVI.member_type_definition_anonymous
`true` if the `name` of the validating type is absent, otherwise `false`.

AttributePSVI.member_type_definition_name
The `name` of the validating type, if it is not absent.

AttributePSVI.member_type_definitions
A sequence of Simple Type Definition components with the same length as the schema actual value, each one an item isomorphic to the validating type of the corresponding item in

the schema actual value.

5.8.8 xsd.AttributeUse

class **xsd.AttributeUse**
An `AttributeUse` class is a utility component which controls the occurrence and defaulting behavior of attribute declarations. It plays the same role for attribute declarations in complex types that particles play for element declarations. *Go to description.*

The `AttributeUse` class provides the following instance attributes (read-only):

AttributeUse.annotations
A sequence of `Annotation` components.

AttributeUse.required
An `xs:boolean` value. Required.

AttributeUse.attribute_declaration
An `AttributeDeclaration` component. Required.

AttributeUse.value_constraint
A `ValueConstraint` property record. Optional.

AttributeUse.inheritable
An `xs:boolean` value. Required.

5.8.9 xsd.Block

class **xsd.Block**
Part of the definition of an element declaration in the schema. Required. *Go to description*.

The `Block` class provides the following **constants**:

```
Block.NONE
Block.EXTENSION
Block.RESTRICTION
Block.SUBSTITUTION
```


5.8.10 xsd.ComplexTypeDefnition

class **xsd.ComplexTypeDefinition**
A `ComplexTypeDefinition` class defines the properties of a complex type through its instance attributes (*listed below*). *Go to description.*

The `ComplexTypeDefinition` class provides the following **instance attributes** (read-only):

ComplexTypeDefinition.annotations
A sequence of `Annotation` components.

ComplexTypeDefinition.name
An `xs:NCName` value. Optional.

ComplexTypeDefinition.target_namespace
An `xs:anyURI` value. Optional.

ComplexTypeDefinition.base_type_definition
A type definition component. Required.

ComplexTypeDefinition.final
A subset of `{extension, restriction}`.

ComplexTypeDefinition.context
Required if `name` instance attribute (*see above*) is absent. Otherwise must be absent. Either an `ElementDeclaration` or a `ComplexTypeDefinition`.

ComplexTypeDefinition.derivation_method
One of `{extension, restriction}`. Required.

ComplexTypeDefinition.abstract
An `xs:boolean` value. Required.

ComplexTypeDefinition.attribute_uses
A set of `AttributeUse` components.

ComplexTypeDefinition.attribute_wildcard
A Wildcard component. Optional.

ComplexTypeDefinition.content_type
A `ContentType` property record. Required.

ComplexTypeDefinition.prohibited_substitutions
A subset of `{extension, restriction}`.

ComplexTypeDefinition.assertions
A sequence of `Assertion` components.

5.8.11 xsd.ContentType

class **xsd.ContentType**
A `ContentType` class specifies the element's content type. *Go to description*

The `ContentType` class provides the following **constants**:

ContentType.EMPTY
ContentType.SIMPLE
ContentType.ELEMENT_ONLY
ContentType.MIXED

The `ContentType` class provides the following **instance attributes** (read-only):

ContentType.variety
One of `{empty, simple, element-only, mixed}`. Required.

ContentType.particle
A `Particle` component. Required if `{variety}` is element-only or mixed, otherwise must be absent.

ContentType.open_content
An `OpenContent` property record. Optional if `{variety}` is element-only or mixed, otherwise must be absent.

ContentType.simple_type_definition
A `SimpleTypeDefinition` component. Required if `{variety}` is simple, otherwise must be absent.

5.8.12 xsd.Defined

class **xsd.Defined**
The `Defined` class represents a keyword member of the set of values allowed for the `disallowed_names` attribute of `NamespaceConstraint`. *Go to description.*

The `Defined` class provides the following **instance method**:

Defined.__str__()

5.8.13 xsd.DerivationMethod

class **xsd.DerivationMethod**
A `DerivationMethod` class provides information about the derivation method. *Go to description*
.

The `DerivationMethod` class provides the following **constants**:

```
DerivationMethod.NONE
DerivationMethod.RESTRICTION
DerivationMethod.EXTENSION
DerivationMethod.LIST
DerivationMethod.UNION
```


5.8.14 xsd.ENTITY

class **xsd.ENTITY**
Represents the ENTITY attribute type of XML. *Go to description*.

The ENTITY class provides the following **instance attributes** (read-only):

ENTITY.value
A string that provides the value of the entity.

5.8.15 xsd.ElementDeclaration

class **xsd.ElementDeclaration**
Element declarations provide for: (i) Local validation of element information item values using a type definition; (ii) Specifying default or fixed values for element information items; (iii) Establishing uniquenesses and reference constraint relationships among the values of related elements and attributes; (iv) Controlling the substitutability of elements through the mechanism of element substitution groups. *Go to description*.

The `ElementDeclaration` class provides the following **instance attributes** (read-only):

ElementDeclaration.annotations
A sequence of `Annotation` components.

ElementDeclaration.name
An `xs:NCName` value. Required.

ElementDeclaration.target_namespace
An `xs:anyURI` value. Optional.

ElementDeclaration.type_definition
A Type Definition component. Required.

ElementDeclaration.type_table
A `TypeTable` property record. Optional.

ElementDeclaration.scope
A `Scope` property record. Required.

ElementDeclaration.value_constraint
A `ValueConstraint` property record. Optional.

ElementDeclaration.nillable
An `xs:boolean` value. Required.

ElementDeclaration.identity_constraint_definitions
A set of `IdentityConstraintDefinition` components.

ElementDeclaration.substitution_group_affiliations
A set of `ElementDeclaration` components.

ElementDeclaration.substitution_group_exclusions
A subset of `{extension, restriction}`.

ElementDeclaration.disallowed_substitutions
A subset of `{substitution, extension, restriction}`.

ElementDeclaration.abstract
An `xs:boolean` value. Required.

5.8.16 xsd.ElementPSVI

class **xsd.ElementPSVI**
If the schema-validity of an element information item has been assessed, then the PSVI properties are returned in instance attributes of the class. *Go to description*.

The `ElementPSVI` class provides the following **instance attributes** (read-only):

ElementPSVI.validity
One of `valid`, `invalid`, or `notKNown`. *Go to description for details*.

ElementPSVI.validation_attempted
One of `full`, `none`, or `partial`. *Go to description for details*.

ElementPSVI.element_declaration
An item isomorphic to the governing declaration component itself.

ElementPSVI.nil
A value of `true` if clause 3.2.3 of Element Locally Valid (Element) is satisfied, otherwise `false`.

ElementPSVI.schema_normalized_value
If a governing type definition is known for an element information item, then in the post-schema-validation infoset the value of the item. See *Element information items*.

ElementPSVI.schema_actual_value
If a governing type definition is known for an element information item, then in the post-schema-validation infoset the value of the item. See *Element information items*.

ElementPSVI.type_definition
If a governing type definition is known for an element information item, then in the post-schema-validation infoset the value of the item. See *Element information items*.

ElementPSVI.type_definition_type
If a governing type definition is known for an element information item, then in the post-schema-validation infoset the value of the item. See *Element information items*.

ElementPSVI.type_definition_namespace
If a governing type definition is known for an element information item, then in the post-schema-validation infoset the value of the item. See *Element information items*.

ElementPSVI.type_definition_anonymous
If a governing type definition is known for an element information item, then in the post-schema-validation infoset the value of the item. See *Element information items*.

ElementPSVI.type_definition_name
If a governing type definition is known for an element information item, then in the post-schema-validation infoset the value of the item. See *Element information items*.

ElementPSVI.member_type_definition
If a governing type definition is known for an element information item, then in the post-schema-validation infoset the value of the item. See *Element information items*.

ElementPSVI.member_type_definition_namespace
If a governing type definition is known for an element information item, then in the post-schema-validation infoset the value of the item. See *Element information items*.

ElementPSVI.member_type_definition_anonymous
If a governing type definition is known for an element information item, then in the post-schema-validation infoset the value of the item. See *Element information items*.

ElementPSVI.member_type_definition_name
If a governing type definition is known for an element information item, then in the post-schema-validation infoset the value of the item. See *Element information items*.

ElementPSVI.member_type_definitions
If a governing type definition is known for an element information item, then in the post-schema-validation infoset the value of the item. See *Element information items*.

ElementPSVI.inherited_attributes
A list of inherited attribute information items. Inheritance is described *here*.

5.8.17 xsd.Final

class **xsd.Final**
A complex type with an empty specification for `Final` can be used as a base type definition for other types derived by either of extension or restriction; the explicit values `extension` and `restriction` prevent further derivations by extension and restriction respectively. If all values are specified, then the complex type is said to be final, because no further derivations are possible. *Go to description*.

The `Final` class provides the following **constants**:

Final.NONE
Final.EXTENSION
Final.RESTRICTION
Final.LIST
Final.UNION

5.8.18 xsd.ID

class **xsd.ID**
Represents the ID attribute type of XML. *Go to description.*

The ID class provides the following **instance attribute** (read-only):

ID.value
A string that gives the value of the ID.

5.8.19 xsd.IDREF

class **xsd.IDREF**
Represents a sequence of ID attribute types of XML. *Go to description*.

The IDREF class provides the following **instance attribute** (read-only):

IDREF.value
A sequence of ID values.

5.8.20 xsd.ID_IDREF_binding

class **xsd.ID_IDREF_binding**
The `ID_IDREF_binding` class represents a binding between `ID` and `IDREF`. *Go to description*

The `ID_IDREF_binding` class provides the following **instance attributes** (read-only):

ID_IDREF_binding.id
ID_IDREF_binding.binding

5.8.21 xsd.ID_IDREF_table

class **xsd.ID_IDREF_table**
The `ID_IDREF_table` class represents a set of `ID-IDREF mappings`. *Go to description*

The `ID_IDREF_table` class provides the following *instance methods*:

ID_IDREF_table.__len__()
ID_IDREF_table.__iter__()

5.8.22 xsd.IdentityConstraintDefinition

class **xsd.IdentityConstraintDefinition**
Identity-constraint definition components provide for uniqueness and reference constraints with respect to the contents of multiple elements and attributes. *Go to description*.

The `IdentityConstraintDefinition` class provides the following **constants**:

IdentityConstraintDefinition.KEY
The identity-constraint definition asserts uniqueness as for `unique`. The constant `key` further asserts that all selected content actually has such tuples.

IdentityConstraintDefinition.KEYREF
The identity-constraint definition asserts a correspondence, with respect to the content identified by `selector`, of the tuples resulting from evaluation of the field's XPath expression (s), with those of the referenced key.

IdentityConstraintDefinition.UNIQUE
The identity-constraint definition asserts uniqueness, with respect to the content identified by `selector`, of the tuples resulting from evaluation of the field's XPath expression(s).

The `IdentityConstraintDefinition` class provides the following **instance attributes** (read-only):

IdentityConstraintDefinition.annotations
A sequence of `Annotation` components.

IdentityConstraintDefinition.name
An `xs:NCName` value. Required.

IdentityConstraintDefinition.target_namespace
An `xs:anyURI` value. Optional.

IdentityConstraintDefinition.identity_constraint_category
One of `{key, keyref, unique}`. Required.

IdentityConstraintDefinition.selector
An `XPathExpression` property record. Required.

IdentityConstraintDefinition.fields
A sequence of `XPathExpression` property records.

IdentityConstraintDefinition.referenced_key
An `IdentityConstraintDefinition` component. Required if `identity_constraint_category` is `keyref`, otherwise (if `identity_constraint_category` is `key` or `unique`) must be absent. If a value is present, its `identity_constraint_category` must be `key` or `unique`.

5.8.23 xsd.Instance

class **xsd.Instance**
The `Instance` class represents the instance document. *Go to description*

The `Instance` class provides the following **instance attributes** (read-only):

Instance.filename
Instance.document
Instance.psvi
Instance.schema

5.8.24 xsd.ModelGroup

class **xsd.ModelGroup**
The `ModelGroup` class specifies a sequential (`sequence`), disjunctive (`choice`) or conjunctive (`all`) interpretation of the `particles` attribute. *Go to description.*

The `ModelGroup` class provides the following **constants**:

ModelGroup.ALL
Determines whether the element information item children validated by the model group must correspond to the specified `particles`. The elements can occur in any order.

ModelGroup.CHOICE
Determines whether the element information item children validated by the model group must correspond to exactly one of the specified `particles`.

ModelGroup.SEQUENCE
Determines whether the element information item children validated by the model group must correspond, in order, to the specified `particles`.

The `ModelGroup` class provides the following **instance attributes** (read-only):

ModelGroup.annotations
A sequence of `Annotation` components.

ModelGroup.compositor
Oe of `{all, choice, sequence}`. Required.

ModelGroup.particles
A sequence of `Particle` components.

5.8.25 xsd.ModelGroupDefinition

class **xsd.ModelGroupDefinition**
A `ModelGroupDefinition` class is identified by its `name` and `target namespace`. Model group identities must be unique within an XSD schema. Model group definitions do not participate in validation, but the `term` of a `Particle` may correspond in whole or in part to a `ModelGroup` from a `ModelGroupDefinition`. The `model_group` instance attribute is the `ModelGroup` for which `ModelGroupDefinition` provides a name. *Go to description*.

The `ModelGroupDefinition` class provides the following **instance attributes** (read-only):

ModelGroupDefinition.annotations
A sequence of `Annotation` components.

ModelGroupDefinition.name
An `xs:NCName` value. Required.

ModelGroupDefinition.target_namespace
An `xs:anyURI` value. Optional.

ModelGroupDefinition.model_group
A `ModelGroup` component. Required.

5.8.26 xsd.NCName

class **xsd.NCName**
The NCName class represents a non-colonized name. *Go to description*.

The NCName class provides the following **instance attribute** (read-only):

NCName.value

5.8.27 xsd.NMTOKEN

class **xsd.NMTOKEN**
The NMTOKEN class represents the NMTOKEN attribute type from XML. *Go to description*.

The NMTOKEN class provides the following **instance attribute** (read-only):

NMTOKEN.value

5.8.28 xsd.NOTATION

class **xsd.NOTATION**
The NOTATION class represents the NOTATION attribute type from XML. *Go to description*.

The NOTATION class provides the following **instance attributes** (read-only):

NOTATION.namespace_name
NOTATION.local_part

5.8.29 xsd.Name

class **xsd.Name**
The `Name` class represents an XML name. *Go to description*.

The `Name` class provides the following **instance attribute** (read-only):

Name.value

5.8.30 xsd.NamespaceBinding

class **xsd.NamespaceBinding**
The `NamespaceBinding` class provides the binding of a `namespace` to a `prefix`. *Go to description*.

The `NamespaceBinding` class provides the following **instance attributes** (read-only):

NamespaceBinding.prefix
NamespaceBinding.namespace

5.8.31 xsd.NamespaceConstraint

class **xsd.NamespaceConstraint**
The `NamespaceConstraint` class provides for validation of attribute and element items that are selected according to the specified constraint. *Go to description*.

The `NamespaceConstraint` class provides the following **constants**:

NamespaceConstraint.ANY
NamespaceConstraint.ENUMERATION
NamespaceConstraint.NOT

The `NamespaceConstraint` class provides the following **instance attributes** (read-only):

NamespaceConstraint.variety
One of `{any, enumeration, not}`. Required.

NamespaceConstraint.namespaces
A set, each of whose members is either an `xs:anyURI` value or the distinguished value `absent`. Required.

NamespaceConstraint.disallowed_names
A set, each of whose members is either an `xs:QName` value, or the keyword `defined`, or the keyword `sibling`. Required.

5.8.32 xsd.NotationDeclaration

class **xsd.NotationDeclaration**
A `NotationDeclaration` class specifies a valid element or attribute value. Notation declarations do not participate in validation as such. They are referenced in the course of validating strings as members of the `NOTATION` simple type. An element or attribute information item with its governing type definition or its validating type derived from the `NOTATION` simple type is valid only if its value was among the enumerations of such simple type. As a consequence such a value is required to be the `name` of a notation declaration. *Go to description*
.

The `NotationDeclaration` class provides the following **instance attributes** (read-only):

`NotationDeclaration.annotations`
A sequence of `Annotation` components.

`NotationDeclaration.name`
An `xs:NCName` value. Required.

`NotationDeclaration.target_namespace`
An `xs:anyURI` value. Optional.

`NotationDeclaration.system_identifier`
An `xs:anyURI` value. Required if `public_identifier` is absent, otherwise optional.

`NotationDeclaration.public_identifier`
A public ID value. Required if `system_identifier` is absent, otherwise optional.

5.8.33 xsd.OpenContent

class **xsd.OpenContent**
An `OpenContent` property record. Optional if `variety` is element-only or mixed, otherwise must be absent. *Go to description*.

The `OpenContent` class provides the following **constants**:

OpenContent.INTERLEAVE
OpenContent.SUFFIX

The `OpenContent` class provides the following **instance attributes** (read-only):

OpenContent.mode
One of `{interleave, suffix}`. Required.

OpenContent.wildcard
A wildcard component. Required.

5.8.34 xsd.PSVI

class **xsd.PSVI**
The PSVI class provides element and attribute schema-validity assessment. *Go to description*.

The PSVI class provides the following **constants**. Also see xsd.ElementPSVI and xsd.AttributePSVI.

PSVI.NOTKNOWN
PSVI.VALID
PSVI.INVALID
PSVI.NONE
PSVI.FULL
PSVI.PARTIAL
PSVI.SIMPLE
PSVI.COMPLEX

The PSVI class provides the following **instance attribute** (read-only):

PSVI.ID_IDREF_table
See xsd.ID_IDREF_table.

The PSVI class provides the following **instance methods**:

PSVI.element()
Provides an element for schema-validity assessment. Also see xsd.ElementPSVI.

PSVI.attribute()
Provides an attribute for schema-validity assessment. Also see xsd.AttributePSVI.

5.8.35 xsd.Particle

class **xsd.Particle**
A `Particle` class contains the components which it either directly contains or indirectly contains. It directly contains the component which is the value of its `term` attribute. It indirectly contains the particles, groups, wildcards, and element declarations which are contained by the value of its `term` property. *Go to description*.

The `Particle` class provides the following **instance attributes** (read-only):

Particle.min_occurs
An `xs:nonNegativeInteger` value. Required.

Particle.max_occurs
Either a positive integer or `unbounded`. Required.

Particle.term
A Term component. Required.

Particle.annotations
A sequence of `Annotation` components.

5.8.36 xsd.QName

class **xsd.QName**
The QName class represents an XML qualified name. *Go to description*

The QName class provides the following **instance attributes** (read-only):

QName.namespace_name
The name of the namespace part of the qualified name.

QName.local_part
The local part of the qualified name.

5.8.37 xsd.Schema

class **xsd.Schema**
The schema class contains a collection of schema components, e.g. type definitions and element declarations, which have a common target namespace. *Go to description*.

The Schema class provides the following **instance attributes** (read-only):

Schema.type_definitions
A set of type definition components. Could be a SimpleTypeDefinition or a ComplexTypeDefinition.

Schema.attribute_declarations
A set of AttributeDeclaration components.

Schema.element_declarations
A set of ElementDeclaration components.

Schema.attribute_group_definitions
A set of AttributeGroupDefinition components.

Schema.model_group_definitions
A set of ModelGroupDefinition components.

Schema.notation_declarations
A set of NotationDeclaration components.

Schema.identity_constraint_definitions
A set of IdentityConstraintDefinition components.

The Schema class provides the following **instance methods**:

Schema.resolve_type_definition()
Provides type definitions.

Schema.resolve_attribute_declaration()
Provides attribute declarations.

Schema.resolve_element_declaration()
Provides element declarations.

Schema.resolve_attribute_group_definition()
Provides attribute group definitions.

Schema.resolve_model_group_definition()
Provides model group definitions.

Schema.resolve_notation_declaration()
Provides notation declarations.

Schema.resolve_identity_constraint_definition()
Provides identity constraint definitions.

5.8.38 xsd.Scope

class **xsd.Scope**
A `Scope` property record. Required. *Go to description*

The `Scope` class provides the following **constants**:

Scope.GLOBAL
Scope.LOCAL

The `Scope` class provides the following **instance attributes** (read-only):

Scope.variety
One of `{global, local}`. Required.

Scope.parent
Either a `ComplexTypeDefinition` or a `AttributeGroupDefinition`. Required if `variety` is `local`, otherwise must be absent.

5.8.39 xsd.Sibling

class **xsd.Sibling**
The `Sibling` class represents a keyword member of the set of values allowed for the `disallowed_names` attribute of `NamespaceConstraint`. *Go to description.*

The `Sibling` class provides the following **instance methods**:

Sibling.__str__()

5.8.40 xsd.SimpleTypeDefinition

class **xsd.SimpleTypeDefinition**
The `SimpleTypeDefinition` class represents simple types identified by their `name` and `target namespace` attributes. For details, *go to description*.

The `SimpleTypeDefinition` class provides the following **constants**:

SimpleTypeDefinition.ATOMIC
SimpleTypeDefinition.LIST
SimpleTypeDefinition.UNION

The `SimpleTypeDefinition` class provides the following **instance attributes** (read-only):

SimpleTypeDefinition.annotations
A sequence of `Annotation` components.

SimpleTypeDefinition.name
An `xs:NCName` value. Optional.

SimpleTypeDefinition.target_namespace
An `xs:anyURI` value. Optional.

SimpleTypeDefinition.context
Required if `name` instance attribute (*see above*) is absent. Otherwise must be absent. Either an `AttributeDeclaration`, `ElementDeclaration`, `ComplexTypeDefinition`, or a `SimpleTypeDefinition`.

SimpleTypeDefinition.base_type_definition
A type definition component. Required.

SimpleTypeDefinition.facets
A set of Constraining Facet components.

SimpleTypeDefinition.final
A subset of `{extension, restriction, list, union}`.

SimpleTypeDefinition.variety
One of `{atomic, list, union}`. Required for all simple type definitions except `xs:anySimpleType`, in which it is absent.

SimpleTypeDefinition.primitive_type_definition
A simple type definition component. With one exception, required if `variety` is atomic, otherwise must be absent. The exception is `xs:anyAtomicType`, whose `primitive_type_definition` is absent. If non-absent, must be a primitive definition.

SimpleTypeDefinition.item_type_definition
A simple type definition component. Required if `variety` is `list`, otherwise must be absent. The value of this property must be a primitive or ordinary simple type definition with `variety=atomic`, or an ordinary simple type definition with `variety=union` whose basic members are all atomic; the value must not itself be a list type (have `variety=list`) or have any basic members which are list types.

SimpleTypeDefinition.member_type_definitions
A sequence of primitive or ordinary SimpleTypeDefinition components. Must be present (but may be empty) if variety=union, otherwise must be absent. The sequence may contain any primitive or ordinary simple type definition, but must not contain any special type definitions.

5.8.41 xsd.TypeAlternative

class **xsd.TypeAlternative**
The `TypeAlternative` class is used by an `ElementDeclaration` to specify a condition (`test`) under which a particular type (`type_definition`) is used as the governing type definition for element information items governed by that `ElementDeclaration`. Each `ElementDeclaration` may have multiple Type Alternatives in its `TypeTable`. *Go to description*

The `TypeAlternative` class provides the following **instance attributes** (read-only):

TypeAlternative.annotations
A sequence of `Annotation` components.

TypeAlternative.test
An `XPathExpression` property record that is used to specify a condition for selecting the governing type declaration of an element declaration. Optional.

TypeAlternative.type_definition
A Type Definition (`xsd.ComplexTypeDefnition` or `xsd.SimpleTypeDefinition`) component. Required.

5.8.42 xsd.TypeTable

class **xsd.TypeTable**
The type definition against which an element information item is validated (its governing type definition) can be different from the declared type definition}. The `TypeTable` property of an `ElementDeclaration`, which governs conditional type assignment, and the `xsi:type` attribute of an element information item can cause the governing type definition and the declared type definition to be different. *Go to description*.

The `TypeTable` class provides the following **instance attributes** (read-only):

TypeTable.alternatives
A sequence of `TypeAlternative` components.

TypeTable.default_type_definition
A `TypeAlternative` component. Required.

5.8.43 xsd.Unbounded

class **xsd.Unbounded**
The `Unbounded` class is a string value. It represents the upper value of the `maxOccurs` property.
Go to description.

The `Unbounded` class provides the following **instance methods**:

Unbounded.__str__()

5.8.44 xsd.ValueConstraint

class **xsd.ValueConstraint**
The `ValueConstraint` class represents a property of the `AttributeUse` class. *Go to description*.

The `ValueConstraint` class provides the following **constants**:

ValueConstraint.DEFAULT
ValueConstraint.FIXED

The `ValueConstraint` class provides the following **instance attributes** (read-only):

ValueConstraint.variety
One of `{default, fixed}`. Required.

ValueConstraint.value
An actual value. Required.

ValueConstraint.lexical_form
A character string. Required.

5.8.45 xsd.XPathExpression

class **xsd.XPathExpression**
To check an assertion, an instance of the XPath 2.0 data model is constructed, in which the element information item being assessed is the (parentless) root node, and elements and attributes are assigned types and values according to XPath 2.0 data model construction rules. When evaluated against this data model instance, `test` evaluates to either `true` or `false`. *Go to description*.

The `XPathExpression` class provides the following **instance attributes** (read-only):

XPathExpression.namespace_bindings
A set of `NamespaceBinding` property records for the XPath expression.

XPathExpression.default_namespace
An `xs:anyURI` value. Optional.

XPathExpression.base_URI
An `xs:anyURI` value. Optional. The base URI for relative paths in the XPath expression.

XPathExpression.expression
An XPath 2.0 expression. Required.

5.8.46 Special Built-in Datatype Objects

The following special built-in datatype objects are available. For a detailed description of the datatype, see its description in the Special Built-in Datatypes and Primitive Datatypes sections of the XML Schema specification.

---

class **xsd.anyAtomicType**
An `anyAtomicType` class represents a restriction of `anySimpleType` and is the base type of the primitive types.

class **xsd.anySimpleType**
An `anySimpleType` class represents a restriction of `anyType` and is the base type of the `anyAtomicType`.

class **xsd.anyURI**
An `anyURI` class represents an Internationalized Resource Identifier (IRI) reference. Its value can be absolute or relative. It has a single read-only instance attribute: **`anyURI.value`**.

5.8.47 String Datatype Objects

The following string datatype objects are available. Each is listed with its read-only instance attributes.

Class	Instance attributes (read-only)
`xsd.language`	`language.value`
`xsd.normalizedString`	`normalizedString.value`
`xsd.string`	`string.value`
`xsd.token`	`token.value`

For a detailed description of the datatype, see its description in the Primitive Datatypes and Other Built-in Datatypes sections of the XML Schema specification.

5.8.48 Boolean Datatype Object

class **xsd.boolean**
A `boolean` object represents an XBRL instance document. It provides the following read-only instance attribute: **`boolean.value`**, which returns a boolean value. For a detailed description of the datatype, see its description in the Primitive Datatypes section of the XML Schema specification.

5.8.49 Number Datatype Objects

The following number datatype objects are available. Each has a single read-only instance attribute: **value**, the lexical representation of each of which is different according to the object.

Class	Instance attribute (read-only)
xsd.byte	byte.value
xsd.decimal	decimal.value
xsd.double	double.value
xsd.float	float.value
xsd.int	int.value
xsd.integer	integer.value
xsd.long	long.value
xsd.negativeInteger	negativeInteger.value
xsd.nonNegativeInteger	nonNegativeInteger.value
xsd.nonPositiveInteger	nonPositiveInteger.value
xsd.positiveInteger	positiveInteger.value
xsd.short	short.value
xsd.unsignedByte	unsignedByte.value
xsd.unsignedInt	unsignedInt.value
xsd.unsignedLong	unsignedLong.value
xsd.unsignedShort	unsignedShort.value

For a detailed description of the datatype, see its definition in the Primitive Datatypes and Other Built-in Datatypes sections of the XML Schema specification.

5.8.50 Duration Datatype Objects

The following duration datatype objects are available. Each is listed with its read-only instance attributes.

Class	Instance attributes (read-only)
`xsd.dayTimeDuration`	`dayTimeDuration.months` `dayTimeDuration.seconds`
`xsd.duration`	`duration.months` `duration.seconds`
`xsd.yearMonthDuration`	`yearMonthDuration.months` `yearMonthDuration.seconds`

For a detailed description of the datatype, see its definition in the Primitive Datatypes and Other Built-in Datatypes sections of the XML Schema specification.

5.8.51 Date and Time Datatype Objects

The following duration datatype objects are available. Each is listed with its read-only instance attributes. If a `value` attribute exists, it is composed of fragments that are available as other attributes of the object. For example: `time.value` consists of the `time.hour`, `time.minute`, `time.second`, and `time.timezoneOffset` fragments.

Class	Instance attributes (read-only)
`xsd.date`	`dayTimeDuration.months` `dayTimeDuration.seconds`
`xsd.dateTime`	`duration.months` `duration.seconds`

`xsd.dateTimeStamp`	`dateTimeStamp.value`
	`dateTimeStamp.year`
	`dateTimeStamp.month`
	`dateTimeStamp.day`
	`dateTimeStamp.hour`
	`dateTimeStamp.minute`
	`dateTimeStamp.second`
	`dateTimeStamp.timezoneOffset`

`xsd.gDay`	`gDay.day` `gDay.timezoneOffset`
`xsd.gMonth`	`gMonth.month` `gMonth.timezoneOffset`

`xsd.gMonthDay`	`gMonthDay.month`
	`gMonthDay.days`
	`gMonthDay.timezoneOffset`

`xsd.gYear`	`gYear.year` `gYear.timezoneOffset`

`xsd.gYearMonth`	`gYearMonth.year`
	`gYearMonth.month`
	`gYearMonth.timezoneOffset`

`xsd.time`	`time.value`
	`time.hour`
	`time.minute`
	`time.second`
	`time.timezoneOffset`

For a detailed description of the datatype, see its definition in the Primitive Datatypes and Other Built-in Datatypes sections of the XML Schema specification.

5.8.52 Binary Datatype Objects

The following binary datatype objects are available. Each is listed with its read-only instance attributes.

Class	Instance attributes (read-only)
xsd.base64Binary	base64Binary.value
xsd.hexBinary	hexBinary.value

For a detailed description of the datatype, see its definition in the Primitive Datatypes section of the XML Schema specification.

5.8.53 Facet Objects

Datatypes derived by restriction may also have constraining facets as allowed by the specification. The following facet objects are available. The table lists facet objects that have only read-only instance attributes. The `xsd.explicitTimezoneFacet` and `xsd.whiteSpaceFacet` objects have constants in addition to their read-only instance attributes and are listed below the table.

For a detailed description of a facet, see its description in the Constraining Facets section of the XML Schema specification. (Clicking a facet object's link in the table below takes you directly to its description.)

Class	Instance attributes (read-only)
xsd.assertionsFacet	assertionsFacet.annotations assertionsFacet.value
xsd.enumerationFacet	enumerationFacet.annotations enumerationFacet.value
xsd.fractionDigitsFacet	fractionDigitsFacet.annotations fractionDigitsFacet.value
xsd.lengthFacet	lengthFacet.annotations lengthFacet.value lengthFacet.fixed
xsd.maxExclusiveFacet	maxExclusiveFacet.annotations maxExclusiveFacet.value maxExclusiveFacet.fixed
xsd.maxInclusiveFacet	maxInclusiveFacet.annotations maxInclusiveFacet.value maxInclusiveFacet.fixed
xsd.maxLengthFacet	maxLengthFacet.annotations maxLengthFacet.value maxLengthFacet.fixed
xsd.minExclusiveFacet	minExclusiveFacet.annotations minExclusiveFacet.value minExclusiveFacet.fixed
xsd.minInclusiveFacet	minInclusiveFacet.annotations minInclusiveFacet.value minInclusiveFacet.fixed
xsd.minLengthFacet	minLengthFacet.annotations minLengthFacet.value minLengthFacet.fixed
xsd.pattern	patternFacet.annotations patternFacet.value
xsd.totalDigitsFacet	totalDigitsFacet.annotations totalDigitsFacet.value totalDigitsFacet.fixed

xsd.explicitTimezoneFacet

- *Constants*:
  `explicitTimezoneFacet.REQUIRED`
  `explicitTimezoneFacet.PROHIBITED`
  `explicitTimezoneFacet.OPTIONAL`
- *Read-only instance attributes*:
  `whiteSpaceFacet.annotations`


```
whiteSpaceFacet.value
whiteSpaceFacet.fixed
```

xsd.whiteSpaceFacet

- *Constants*:
  ```
  whiteSpaceFacet.PRESERVE
  whiteSpaceFacet.REPLACE
  whiteSpaceFacet.COLLAPSE
  ```
- *Read-only instance attributes*:
  ```
  whiteSpaceFacet.annotations
  whiteSpaceFacet.value
  whiteSpaceFacet.fixed
  ```

5.9 Python XBRL API

The **xbrl** module provides a Python interface to the C++ implementation of the XBRL data model layer.

Available types
The following types are available. They are described in detail in the sub-sections of this section.

class xbrl.Instance
The `Instance` class represents an XBRL instance document.

class xbrl.DTS
The `DTS` class represents an XBRL Discovery Taxonomy Set (DTD).

class xbrl.Concept
The `Concept` class represents an XBRL concept in the DTS.

class xbrl.Fact
The`NFact` class represents a fact element in an XBRL instance document.

class xbrl.Fraction
The `Fraction` class represents the fraction value of an XBRL fact item of type.

class xbrl.Context
The `Context` class represents an XBRL context.

class xbrl.Unit
The `Unit` class represents an XBRL unit.

class xbrl.Entity
The `Entity` class represents the entity part of an XBRL context.

class xbrl.EntityIdentifier
The `EntityIdentifier` class represents the entity identifier part of an XBRL context.

class xbrl.Period
The `Period` class represents the period part of an XBRL context.

5.9.1 xbrl.Instance

class **xbrl.Instance**
An Instance object represents an XBRL instance document.

The Instance class provides the following **instance attributes** (read-only):

Instance.filename
Returns a string with the URL of the XBRL instance document.

Instance.document
Returns an xml.Document object which represents the XML document information item of the XBRL instance document.

Instance.psvi
Returns an xsd.PSVI object which represents the XML Schema PSVI information.

Instance.dts
Returns a DTS object which represents the XBRL Discoverable Taxonomy Set.

Instance.contexts
Returns a list of Context objects which represents all XBRL contexts present in the instance document.

Instance.units
Returns a list of Unit objects which represents all XBRL units present in the instance document.

Instance.facts
Returns a list of Fact objects which represents all XBRL facts present in the instance document. This also includes facts which are children of other tuples.

Instance.non_nil_facts
Returns a list of Fact objects which represents all non-nil XBRL facts present in the instance document. This also includes facts which are children of other tuples.

Instance.items
Returns a list of Fact objects which represents all top-level XBRL item facts present in the instance document. Facts that are children of other tuples are not included.

Instance.tuples
Returns a list of Fact objects which represents all top-level XBRL tuple facts present in the instance document. Facts that are children of other tuples are not included.

5.9.2 xbrl.DTS

class **xbrl.DTS**
A DTS class represents an XBRL Discoverable Taxonomy Set.

The DTS class provides the following **instance attributes** (read-only):

DTS.schema
Returns an xsd.Schema object which represents the XML Schema component of the underlying DTS.

DTS.linkbases
Returns a list of xml.Document objects which represent all XBRL linkbases in the DTS.

DTS.items
Returns an iterator over Concept objects which represent all XBRL concepts in the DTS that are in the substitution group of xbrli:item.

DTS.tuples
Returns an iterator over Concept objects which represent all XBRL concepts in the DTS that are in the substitution group of xbrli:tuple.

DTS.hypercubes
Returns an iterator over Concept objects which represent all XBRL concepts in the DTS that are in the substitution group of xbrldt:hypercubeItem.

DTS.dimensions
Returns an iterator over Concept objects which represent all XBRL concepts in the DTS that are in the substitution group of xbrldt:dimensionItem.

The DTS class provides the following **instance methods**:

DTS.resolve_concept(qname)
Returns a Concept object which represents the XBRL concept with the given XML qualified name qname in the DTS. If there is no such concept, this method returns None.

5.9.3 xbrl.Concept

class **xbrl.Concept**
A Concept class represents an XBRL concept in the DTS.

The Concept class provides the following **constants**:

Concept.ITEM
Denotes that the concept is in the substitution group of the xbrli:item XBRL concept.

Concept.TUPLE
Denotes that the concept is in the substitution group of the xbrli:tuple XBRL concept.

Concept.HYPERCUBE
Denotes that the concept is in the substitution group of the xbrli:hypercube XBRL concept.

Concept.DIMENSION_EXPLICIT
Denotes that the concept is in the substitution group of the xbrli:dimension XBRL concept and it does not have the attribute xbrldt:typedDomainRef set.

Concept.DIMENSION_TYPED
Denotes that the concept is in the substitution group of the xbrli:dimension XBRL concept and it does have the attribute xbrldt:typedDomainRef set.

Concept.DEBIT
Denotes that the concept has the attribute xbrli:balance set to debit.

Concept.CREDIT
Denotes that the concept has the attribute xbrli:balance set to credit.

Concept.INSTANT
Denotes that the concept has the attribute xbrli:periodType set to instant.

Concept.DURATION
Denotes that the concept has the attribute xbrli:periodType set to duration.

Concept.DECIMAL_ITEM_TYPE
Denotes that the concept's type definition is xbrli:decimalItemType or a type definition derived from it.

Concept.FLOAT_ITEM_TYPE
Denotes that the concept's type definition is xbrli:floatItemType or a type definition derived from it.

Concept.DOUBLE_ITEM_TYPE
Denotes that the concept's type definition is xbrli:doubleItemType or a type definition derived from it.

Concept.INTEGER_ITEM_TYPE
Denotes that the concept's type definition is xbrli:integerItemType or a type definition derived from it.

Concept.NON_POSITIVE_INTEGER_ITEM_TYPE
Denotes that the concept's type definition is xbrli:nonPositiveIntegerItemType or a type definition derived from it.

Concept.NEGATIVE_INTEGER_ITEM_TYPE
Denotes that the concept's type definition is `xbrli:negativeIntegerItemType` or a type definition derived from it.

Concept.LONG_ITEM_TYPE
Denotes that the concept's type definition is `xbrli:longItemType` or a type definition derived from it.

Concept.INT_ITEM_TYPE
Denotes that the concept's type definition is `xbrli:intItemType` or a type definition derived from it.

Concept.SHORT_ITEM_TYPE
Denotes that the concept's type definition is `xbrli:shortItemType` or a type definition derived from it.

Concept.BYTE_ITEM_TYPE
Denotes that the concept's type definition is `xbrli:byteItemType` or a type definition derived from it.

Concept.NON_NEGATIVE_INTEGER_ITEM_TYPE
Denotes that the concept's type definition is `xbrli:nonNegativeIntegerItemType` or a type definition derived from it.

Concept.UNSIGNED_LONG_ITEM_TYPE
Denotes that the concept's type definition is `xbrli:unsignedLongItemType` or a type definition derived from it.

Concept.UNSIGNED_INT_ITEM_TYPE
Denotes that the concept's type definition is `xbrli:unsignedIntItemType` or a type definition derived from it.

Concept.UNSIGNED_SHORT_ITEM_TYPE
Denotes that the concept's type definition is `xbrli:unsignedShortItemType` or a type definition derived from it.

Concept.UNSIGNED_BYTE_ITEM_TYPE
Denotes that the concept's type definition is `xbrli:unsignedByteItemType` or a type definition derived from it.

Concept.POSITIVE_INTEGER_ITEM_TYPE
Denotes that the concept's type definition is `xbrli:positiveIntegerItemType` or a type definition derived from it.

Concept.MONETARY_ITEM_TYPE
Denotes that the concept's type definition is `xbrli:monetaryItemType` or a type definition derived from it.

Concept.SHARES_ITEM_TYPE
Denotes that the concept's type definition is `xbrli:sharesItemType` or a type definition derived from it.

Concept.PURE_ITEM_TYPE
Denotes that the concept's type definition is `xbrli:pureItemType` or a type definition derived from it.

Concept.FRACTION_ITEM_TYPE

Denotes that the concept's type definition is `xbrli:fractionItemType` or a type definition derived from it.

Concept.STRING_ITEM_TYPE
Denotes that the concept's type definition is `xbrli:stringItemType` or a type definition derived from it.

Concept.BOOLEAN_ITEM_TYPE
Denotes that the concept's type definition is `xbrli:booleanItemType` or a type definition derived from it.

Concept.HEXBINARY_ITEM_TYPE
Denotes that the concept's type definition is `xbrli:hexBinaryItemType` or a type definition derived from it.

Concept.BASE64BINARY_ITEM_TYPE
Denotes that the concept's type definition is `xbrli:base64BinaryItemType` or a type definition derived from it.

Concept.ANYURI_ITEM_TYPE
Denotes that the concept's type definition is `xbrli:anyURIItemType` or a type definition derived from it.

Concept.QNAME_ITEM_TYPE
Denotes that the concept's type definition is `xbrli:QNameItemType` or a type definition derived from it.

Concept.DURATION_ITEM_TYPE
Denotes that the concept's type definition is `xbrli:durationItemType` or a type definition derived from it.

Concept.DATETIME_ITEM_TYPE
Denotes that the concept's type definition is `xbrli:dateItemItemType` or a type definition derived from it.

Concept.TIME_ITEM_TYPE
Denotes that the concept's type definition is `xbrli:timeItemType` or a type definition derived from it.

Concept.DATE_ITEM_TYPE
Denotes that the concept's type definition is `xbrli:dateItemType` or a type definition derived from it.

Concept.GYEARMONTH_ITEM_TYPE
Denotes that the concept's type definition is `xbrli:gYearMonthItemType` or a type definition derived from it.

Concept.GYEAR_ITEM_TYPE
Denotes that the concept's type definition is `xbrli:gYearItemType` or a type definition derived from it.

Concept.GMONTHDAY_ITEM_TYPE
Denotes that the concept's type definition is `xbrli:gMonthDayItemType` or a type definition derived from it.

Concept.GDAY_ITEM_TYPE
Denotes that the concept's type definition is `xbrli:gDayItemType` or a type definition derived from it.

Concept.GMONTH_ITEM_TYPE
Denotes that the concept's type definition is `xbrli:gMonthItemType` or a type definition derived from it.

Concept.NORMALIZED_STRING_ITEM_TYPE
Denotes that the concept's type definition is `xbrli:normalizedStringItemType` or a type definition derived from it.

Concept.TOKEN_ITEM_TYPE
Denotes that the concept's type definition is `xbrli:tokenItemType` or a type definition derived from it.

Concept.LANGUAGE_ITEM_TYPE
Denotes that the concept's type definition is `xbrli:languageItemType` or a type definition derived from it.

Concept.NAME_ITEM_TYPE
Denotes that the concept's type definition is `xbrli:NameItemType` or a type definition derived from it.

Concept.NCNAME_ITEM_TYPE
Denotes that the concept's type definition is `xbrli:NCNameItemType` or a type definition derived from it.

The `Concept` class provides the following **instance attributes** (read-only):

Concept.element
Returns an `xml.Element` object which represents the XML element information item of the concept's schema element declaration.

Concept.element_declaration
Returns an `xsd.ElementDeclaration` object which represents the concept's schema element declaration.

Concept.qname
Returns an `xml.QName` object which represents the XML qualified name of the concept.

Concept.balance
Returns `DEBIT` or `CREDIT` if the concept's schema element declaration contains an `xbrli:balance` attribute. Otherwise this attribute is `None`.

Concept.period_type
Returns `INSTANT` or `DURATION` if the concept's schema element declaration contains an `xbrli:periodType` attribute. Otherwise this attribute is `None`.

Concept.concept_type
Returns one of `ITEM`, `TUPLE`, `HYPERCUBE`, `DIMENSION_EXPLICIT`, or `DIMENSION_TYPED` depending on the substitution group affiliation of the concept's schema element declaration.

Concept.item_type
Returns the built-in XBRL type from which the concept's type definition is derived from. Examples are, for example, `STRING_ITEM_TYPE`, `MONETARY_ITEM_TYPE`, `SHARES_ITEM_TYPE`.

Concept.label_elements
Returns a list of `xml.Element` objects representing all label resources that are associated

with this concept.

Concept.reference_elements
Returns a list of xml.Element objects representing all reference resources that are associated with this concept.

The **Concept** class provides the following **instance methods**:

Concept.is_numeric()
Returns True if the concept's type definition is derived from one of the built-in XBRL numeric types.

Concept.is_non_numeric()
Returns True if the concept's type definition is derived from one of the built-in XBRL non-numeric types.

Concept.is_fraction()
Returns True if the concept's type definition is derived from xbrli:fractionItemType.

Concept.is_abstract()
Returns True if the concept's schema element declaration has the abstract component property set to True.

Concept.is_nillable()
Returns True if the concept's schema element declaration has the nillable component property set to True.

5.9.4 xbrl.Fact

class **xbrl.Fact**
A Fact class represents a fact element in an XBRL instance document.

The Fact class provides the following **instance attributes** (read-only):

Fact.element
Returns an xml.Element object which represents the XML element information item of the XBRL fact.

Fact.qname
Returns an xml.QName object which represents the XML qualified name of the XBRL fact.

Fact.concept
Returns a Concept object which represents the XBRL concept associated with this XBRL fact.

Fact.context
Returns a Context object which represents the XBRL context associated with this XBRL fact.

Fact.unit
Returns a Unit object which represents the XBRL unit associated with this XBRL fact.

Fact.nil
Returns `True` if the XBRL fact's element information item has the `xsi:nil` attribute set to `True`.

Fact.decimals
Returns a string containing the value of the `decimals` attribute on the fact's element information item. If no decimals attribute has been defined, then the attribute is `None`.

Fact.precision
Returns a string containing the value of the `precision` attribute on the fact's element information item. If no `precision` attribute has been defined, then the attribute is `None`.

Fact.normalized_value
Returns the XBRL fact's schema normalized value as a string.

Fact.effective_numeric_value
Returns a `decimal.Decimal` object which represents the XBRL fact's effective numeric value after rounding with the information provided by the `decimals` or `precision` attribute. If the fact's type definition is not derived from a built-in XBRL numeric type definition, then the attribute returns `NaN`.

Fact.fraction_value
Returns a Fraction object which represents the fraction value denoted by the XBRL fact. If the fact's type definition is not derived from the built-in XBRL type definition `xbrli:fractionItemType`, then the attribute returns `None`.

Fact.child_facts
Returns a list of Fact objects representing all XBRL facts which are direct children of this fact.

Fact.child_items
Returns a list of Fact objects representing only XBRL item facts which are direct children of

this fact.

Fact.child_tuples
Returns a list of Fact objects representing only XBRL tuple facts which are direct children of this fact.

Fact.footnote_elements
Returns a list of xml.Element objects representing all footnote resources that are associated with this fact.

The Fact class provides the following **instance methods**:

Fact.is_item()
Returns True if the fact is an XBRL item fact.

Fact.is_tuple()
Returns True if the fact is an XBRL tuple fact.

Fact.location_aspect()
Returns an xml.Element object which represents the location aspect of this XBRL fact.

Fact.concept_aspect()
Returns a Concept object which represents the concept aspect of this XBRL fact.

Fact.entity_identifier_aspect()
Returns an EntityIdentifier object which represents the entity identifier aspect of this XBRL fact. If the XBRL fact has no entity identifier aspect, then this attribute is None.

Fact.period_aspect()
Returns a Period object which represents the period aspect of this XBRL fact. If the XBRL fact has no period aspect, then this attribute is None.

Fact.unit_aspect()
Returns a Unit object which represents the unit aspect of this XBRL fact. If the XBRL fact has no unit aspect, then this attribute is None.

Fact.complete_segment_aspect()
Returns an xml.Element object which represents the complete segment aspect of this XBRL fact. If the XBRL fact has no complete segment aspect, then this attribute is None.

Fact.complete_scenario_aspect()
Returns an xml.Element object which represents the complete scenario aspect of this XBRL fact. If the XBRL fact has no complete scenario aspect, then this attribute is None.

Fact.non_xdt_segment_aspect()
Returns a list of xml.Element objects which represent the non-XDT segment aspect of this XBRL fact. If the XBRL fact has no non-XDT segment aspect, then this attribute is None.

Fact.non_xdt_scenario_aspect()
Returns a list of xml.Element objects which represent the non-XDT scenario aspect of this XBRL fact. If the XBRL fact has no non-XDT scenario aspect, then this attribute is None.

Fact.explicit_dimension_aspect(qname)
Returns an xml.QName object which represents the given *qname* dimension aspect of this XBRL fact (which is the domain member QName). If the XBRL fact has no such dimension

aspect, then this attribute is `None`. The argument *qname* must be an object of class `xml.QName`.

`Fact.typed_dimension_aspect(qname)`
Returns an `xml.Element` object which represents the given *qname* `dimension` aspect of this XBRL fact (which is the typed domain element). If the XBRL fact has no such `dimension` aspect, then this attribute is `None`. The argument *qname* must be an object of class `xml.QName`.

5.9.5 xbrl.Fraction

class **xbrl.Fraction(numerator, denominator)**
A Fraction object represents the fraction value of an XBRL fact item of type fractionItemType. It constructs an object of class Fraction. All arguments are required.

The Fraction class provides the following instance attributes:

Fraction.numerator
Returns a decimal.Decimal object which represents the numerator part of the fraction value.

Fraction.denominator
Returns a decimal.Decimal object which represents the denominator part of the fraction value.

5.9.6 xbrl.Context

class **xbrl.Context**
A Context class represents an XBRL context in the instance document.

The Context class provides the following **instance attributes** (read-only):

Context.element
Returns an xml.Element object which represents the XML element information item of the XBRL context.

Context.id
Returns a string with the value of the id attribute of the XBRL context.

Context.period
Returns an xbrl.Period object which represents the period part of the XBRL context.

Context.entity
Returns an xbrl.Entity object which represents the entity part of the XBRL context.

Context.scenario_element
Returns an xml.Element object which represents the scenario child XML element information item of the XBRL context.

5.9.7 xbrl.Unit

class **xbrl.Unit**
A Unit class represents an XBRL unit in the instance document.

The Unit class provides the following **instance attributes** (read-only):

Unit.element
Returns an xml.Element object which represents the XML element information item of the XBRL unit.

Unit.id
Returns a string with the value of the id attribute of the XBRL unit.

Unit.numerators
Returns a list of xml.QName objects which represents the QNames in the numerator of the XBRL unit.

Unit.denominators
Returns a list of xml.QName objects which represents the QNames in the denominator of the XBRL unit.

5.9.8 xbrl.Entity

class **xbrl.Entity**
An Entity class represents the entity part of an XBRL context in the instance document.

The Entity class provides the following **instance attributes** (read-only):

Entity.element
Returns an xml.Element object which represents the entity child XML element information item of the XBRL context.

Entity.identifier
Returns an xbrl.EntityIdentifier object which represents the entity identifier part of the XBRL context.

Entity.segment_element
Returns an xml.Element object which represents the segment child XML element information item of the XBRL context.

5.9.9 xbrl.EntityIdentifier

class **xbrl.EntityIdentifier**
An EntityIdentifier class represents the entity identifier part of an XBRL context in the instance document.

The EntityIdentifier class provides the following **instance attributes** (read-only):

EntityIdentifier.element
Returns a xml.Element object which represents the identifier child XML element information item of the XBRL context.

EntityIdentifier.value
Returns a string with the text value of the identifier child XML element information item of the XBRL context.

EntityIdentifier.scheme
Returns a string with the text value of the scheme attribute of the identifier child XML element information item.

5.9.10 xbrl.Period

class **xbrl.Period**
A Period class represents the period part of an XBRL context in the instance document.

The Period class provides the following **constants**:

Period.INSTANT
Denotes that the period contains an `instant` child XML information item.

Period.DURATION
Denotes that the period contains `startDate` and `endDate` child XML information items.

Period.FOREVER
Denotes that the period contains a `forever` child XML information item.

The Period class provides the following **instance attributes** (read-only):

Period.element
Returns an `xml.Element` object which represents the `period` child XML element information item of the XBRL context.

Period.period_type
Returns INSTANT, DURATION or FOREVER depending on the period's child XML element information items.

Period.instant
Returns a string with the text value of the `instant` child XML element information item. If the period doesn't have an `instant` child XML element information item, the attribute is `None`.

Period.start_date
Returns a string with the text value of the `startDate` child XML element information item. If the period doesn't have an `startDate` child XML element information item, the attribute is `None`.

Period.end_date
Returns a string with the text value of the `endDate` child XML element information item. If the period doesn't have an `endDate` child XML element information item, the attribute is `None`.

Period.effective_instant
Returns a `datetime.datetime` object with effective date and time calculated from the value of the `instant` child XML element information item. If the period doesn't have an `instant` child XML element information item, the attribute is `None`.

Period.effective_start_date
Returns a `datetime.datetime` object with effective date and time calculated from the value of the `startDate` child XML element information item. If the period doesn't have a `startDate` child XML element information item, the attribute is `None`.

Period.effective_end_date
Returns a `datetime.datetime` object with effective date and time calculated from the value of the `endDate` child XML element information item. If the period doesn't have an `endDate` child XML element information item, the attribute is `None`.

Chapter 6

Java Interface

6 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:

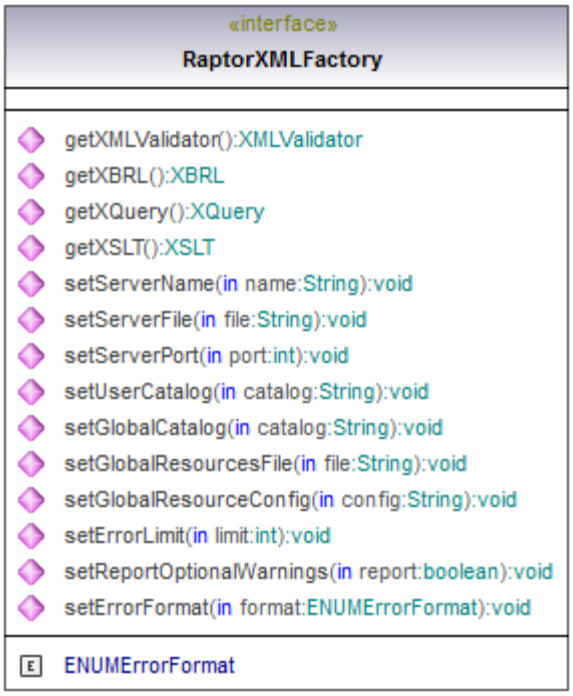
```
public interface RaptorXMLFactory
{
   public XMLValidator getXMLValidator();
   public XSLT getXBRL();
   public XQuery getXQuery();
   public XSLT getXSLT();
   public void setServerName( String name ) throws RaptorXMLException;
   public void setServerFile( String file ) throws RaptorXMLException;
   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.

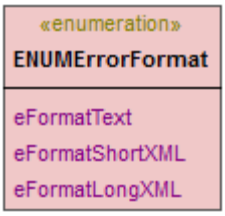
- RaptorXMLFactory
  Creates new RaptorXML COM server object instance via native call, and provides access to RaptorXML engines.
- XMLValidator
  Class holding XMLValidator.
- XBRL
  Class holding the XBRL Validator
- XSLT
  Class holding the XSLT 1.0, 2.0, 3.0 Engines.
- XQuery
  Class holding the XQuery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

6.1 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.				

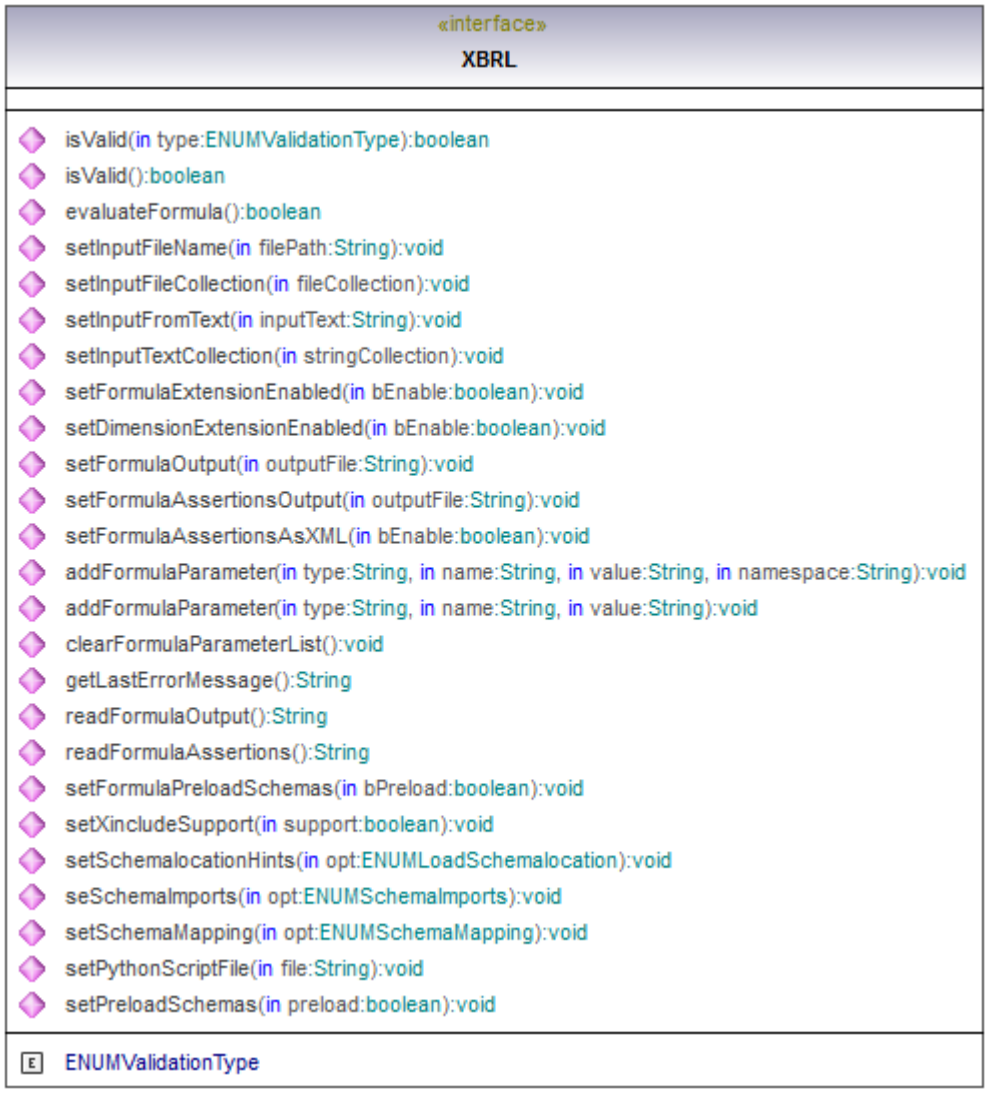
Operation **RaptorXMLFactory::setUserCatalog**

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.				

6.2 RaptorXMLJava - XBRL

Interface **XBRL**

diagram



typedElements

Interface **RaptorXMLFactory** Operation **getXBRL**

documentation

Validates the supplied XML or XBRL (Note: XBRL only with RaptorXML+XBRL) document against internal or external DTDs or XML Schemas.

Checks well-formedness of XML, DTD, XML Schema, XSLT, and XQuery documents.

Validates XBRL taxonomies, and XBRL documents against XBRL taxonomies (only RaptorXML+XBRL).

Operation **XBRL::addFormulaParameter**

parameter	name	direction	type	multiplicity	default

r	**type**	**in**	**String**
	name	**in**	**String**
	value	**in**	**String**
	namespace	**in**	**String**
	return	**return**	**void**
documentation	Adds a parameter used in the formula evaluation process. Parameters: 'type' holds the parameter data type - of type string. 'name' holds the parameter name - of type string. 'value' holds the parameter value - of type string. 'namespace' holds the parameter namespace - of type string.		

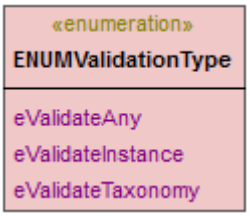
Operation **XBRL::addFormulaParameter**

parameter	name	direction	type	multiplicity	default
	type	**in**	**String**		
	name	**in**	**String**		
	value	**in**	**String**		
	return	**return**	**void**		
documentation	Adds a parameter used in the formula evaluation process. Parameters: 'type' holds the parameter data type - of type string. 'name' holds the parameter name - of type string. 'value' holds the parameter value - of type string.				

Operation **XBRL::clearFormulaParameterList**

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

Enumeration **XBRL::ENUMValidationType**

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

EnumerationLiteral **XBRL::ENUMValidationType::eValidateAny**

documentation	Validates any one XBRL (instance or taxonomy) document. Note that this command is also used to validate XML, DTD, XSD, XSLT, or XQuery documents; the type of document submitted is detected automatically.

EnumerationLiteral **XBRL::ENUMValidationType::eValidateInstance**

documentation	Validates an XBRL instance document (.xbrl extension).

EnumerationLiteral **XBRL::ENUMValidationType::eValidateTaxonomy**

documentation	Validates an XBRL taxonomy (schema) document (.xsd extension).

Operation **XBRL::evaluateFormula**

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

documentation	Evaluates XBRL formulas in an XBRL instance file. Returns: 'true' on success, and false on false on failure. If an error occurs during execution, use the getLastErrorMessage operation to access additional information. Raises a RaptorXMLException when an error occurs.

Operation **XBRL::getLastErrorMessage**

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

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

Operation **XBRL::isValid**

parameter	name	direction	type	multiplicity	default
	type	**in**	**ENUMValidationType**		
	return	**return**	**boolean**		

documentation	Result of the XBRL validation against the taxonomy 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 **XBRL::isValid**

parameter	name	direction	type	multiplicity	default
	return	**return**	**boolean**		
documentation	Result of the XBRL validation against the taxonomy. 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 **XBRL::readFormulaAssertions**

parameter	name	direction	type	multiplicity	default
	return	**return**	**String**		
documentation	Reads the formula assertion evaluation from the specified file. Parameters: none.				

Operation **XBRL::readFormulaOutput**

parameter	name	direction	type	multiplicity	default
	return	**return**	**String**		
documentation	Reads the output of the formula assertion evaluation from the specified file. Parameters: none.				

Operation **XBRL::setSchemaImports**

parameter	name	direction	type	multiplicity	default
	opt	**in**	**ENUMSchemaImports**		
	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 **XBRL::setDimensionExtensionEnabled**

parameter	name	direction	type	multiplicity	default
	bEnable	**in**	**boolean**		
	return	**return**	**void**		
documentation	Enables XBRL Dimension extension validation. Parameters: 'bEnable' holds the boolean value - of type boolean. Values: 'true' enables the dimension extension validaton, 'false' disables it.				

Operation **XBRL::setFormulaAssertionsAsXML**

parameter	name	direction	type	multiplicity	default
	bEnable	**in**	**boolean**		
	return	**return**	**void**		
documentation	Enables XML formatting of the assertion file when RaptorXML is run with assertions enabled. Parameters: 'bEnable' holds the boolean value - of type boolean. Values: 'true' enables XML output, 'false' generates JSON output.				

Operation **XBRL::setFormulaAssertionsOutput**

parameter	name	direction	type	multiplicity	default
	outputFile	**in**	**String**		
	return	**return**	**void**		
documentation	Sets the location of the formula assertion output file. Parameters: 'outputFile' holds the full path of the output file - of type string.				

Operation **XBRL::setFormulaExtensionEnabled**

parameter	name	direction	type	multiplicity	default
	bEnable	**in**	**boolean**		
	return	**return**	**void**		
documentation	Enables XBRL formula extensions validation. Parameters: 'bEnable' holds the boolean value - of type boolean. Values: 'true' enables formula extension validation, 'false' disables it.				

Operation **XBRL::setFormulaOutput**

parameter	name	direction	type	multiplicity	default
	outputFile	**in**	**String**		
	return	**return**	**void**		
documentation	Sets the location for the output of the XBRL formula evaluation file. Parameters: 'outputFile' is the full path of the formula output file.				

Operation **XBRL::setFormulaPreloadSchemas**

parameter	name	direction	type	multiplicity	default
	bPreload	**in**	**boolean**		
	return	**return**	**void**		
documentation	Defines if the XBRL formula schemas will be preloaded. Parameters: 'bPreload' holds the boolean value - of type boolean. Values: 'true' preloads schemas of the XBRL Formula 1.0 specification. Default is 'false'.				

Operation **XBRL::setInputFileCollection**

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

Operation **XBRL::setInputFileName**

parameter	name	direction	type	multiplicity	default
	filePath	**in**	**String**		
	return	**return**	**void**		
documentation	Sets the file name for the XBRL input instance file. Parameters: 'filePath' is a string containing the absolute URL (or base location) of the input file.				

Operation **XBRL::setInputFromText**

parameter	name	direction	type	multiplicity	default
	inputText	**in**	**String**		
	return	**return**	**void**		
documentation	Supplies the contents of the XBRL input document as text. Parameters: 'inputText' contains the XBRL data - of type string.				

Operation **XBRL::setInputTextCollection**

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

Operation **XBRL::setPreloadSchemas**

parameter	name	direction	type	multiplicity	default
	preload	**in**	**boolean**		
	return	**return**	**void**		
documentation	Defines if the XBRL 2.1 schemas will be preloaded. Parameters: 'preload' holds the boolean value - of type boolean. Values: 'true' preloads schemas of the XBRL schemas Default is 'false'.				

Operation **XBRL::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 **XBRL::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 **XBRL::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.				

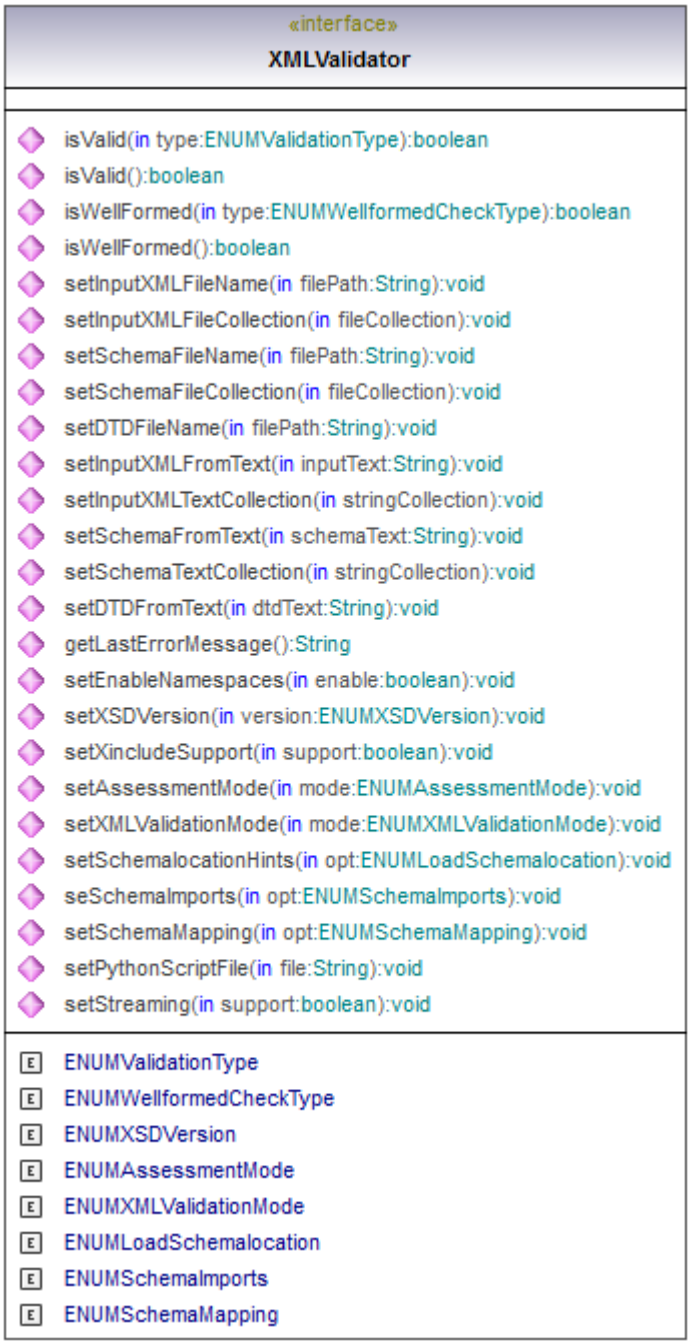
Operation **XBRL::setXincludeSupport**

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.				

6.3 RaptorXMLJava - XMLValidator

Interface **XMLValidator**

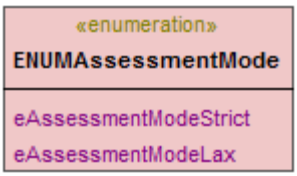
diagram



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.

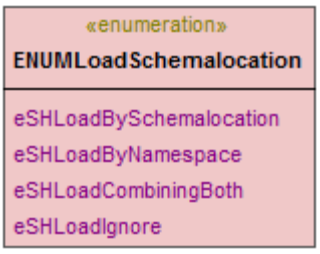
EnumerationLiteral **XMLValidator::ENUMAssessmentMode::eAssessmentModeLax**

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.

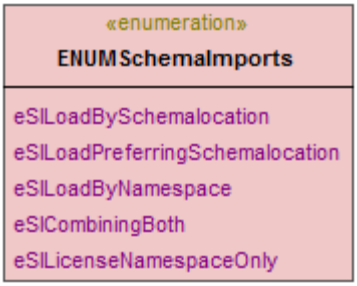
EnumerationLiteral **XMLValidator::ENUMLoadSchemalocation::eSHLoadCombiningBoth**

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.

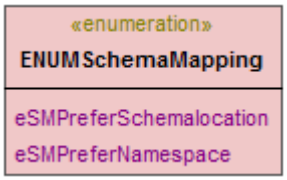
EnumerationLiteral **XMLValidator::ENUMSchemaImports::eSILoadBySchemalocation**

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.

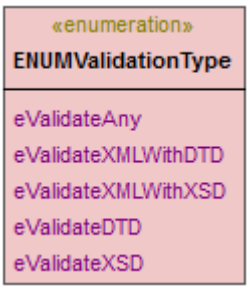
EnumerationLiteral **XMLValidator::ENUMSchemaMapping::eSMPreferNamespace**

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.

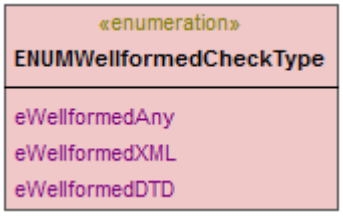
EnumerationLiteral **XMLValidator::ENUMValidationType::eValidateXMLWithXSD**

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.

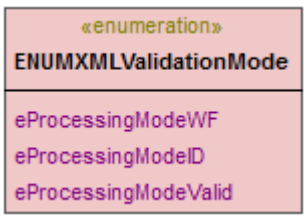
EnumerationLiteral **XMLValidator::ENUMWellformedCheckType::eWellformedDTD**

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

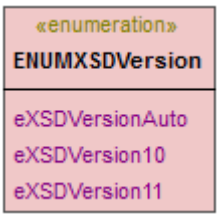
EnumerationLiteral **XMLValidator::ENUMXMLValidationMode::eProcessingModeValid**

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.				

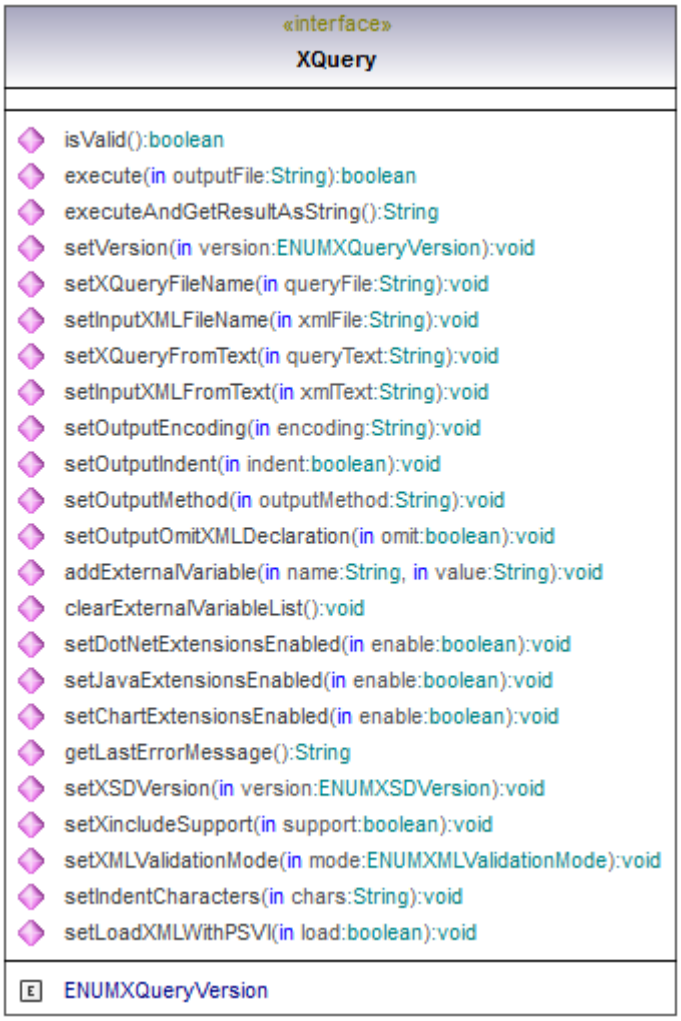
Operation **XMLValidator::setXSDVersion**

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.				

6.4 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**		

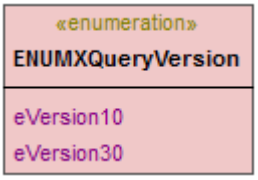
documentation	Adds the name and value of a new external variable. Parameters: 'name' is the variable name, and is a valid QName - of type string. 'value' is the value of the variable - of type string. Each external variable and its value is to be specified in a separate call to the method. Variables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

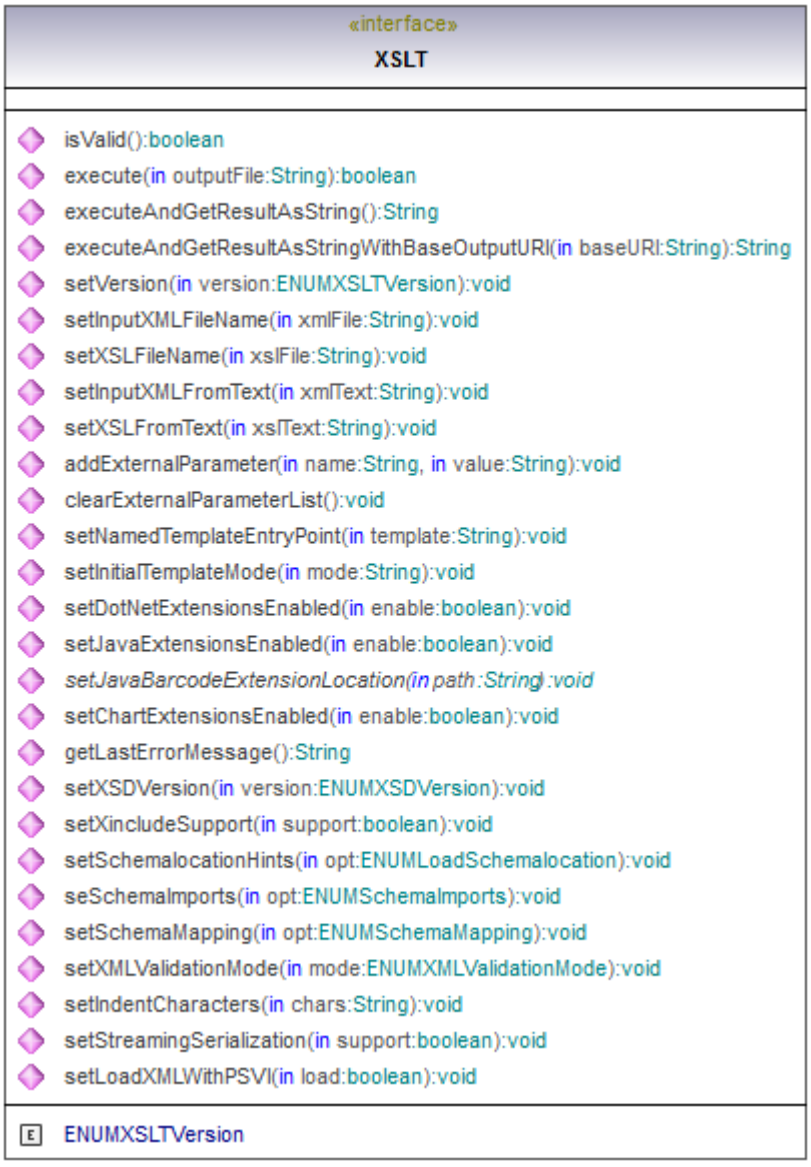
parameter	name	direction	type	multiplicity	default
	version	**in**	**ENUMXSDVersion**		
	return	**return**	**void**		
documentation	Defines the XML Schema version that the document will be validated against. Parameters: 'version' holds the XML Schema version set by the EnumXSDVersion literal - of				

	type EnumXSDVersion.

6.5 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.

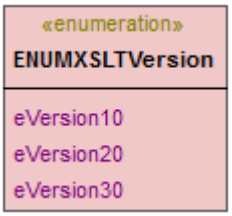
Operation **XSLT::addExternalParameter**

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

6.6 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 7

COM / .NET Interface

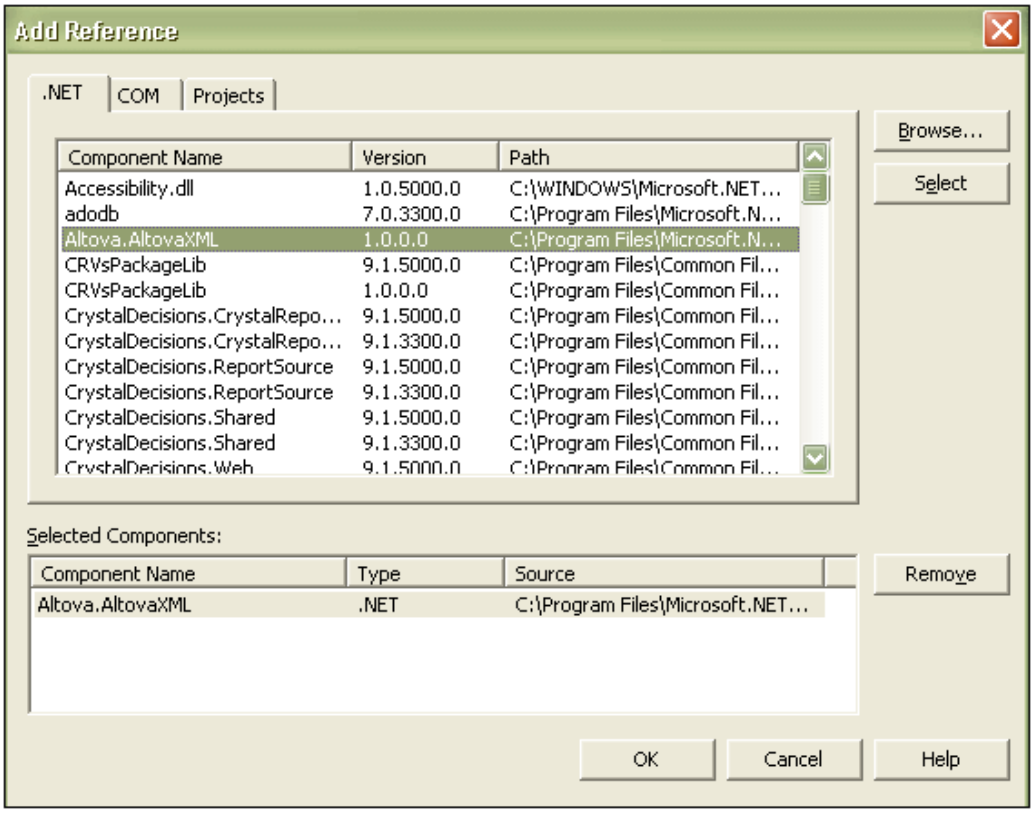
7 COM / .NET Interface

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

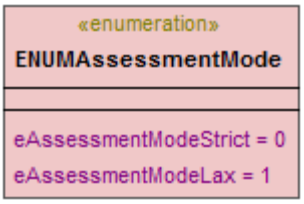
to the `RaptorXML_COM.exe` must be entered. See Registering RaptorXML as a COM Server Object for more details.)

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.

7.1 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.

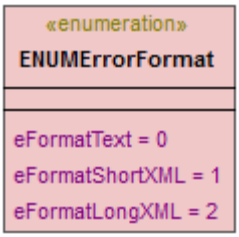
EnumerationLiteral **ENUMAssessmentMode::eAssessmentModeStrict**

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

7.2 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**

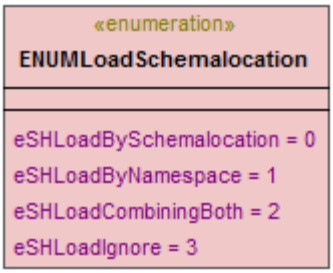
documentation	Sets the error output format to Short XML. This XML output format is an abbreviated form of the XML Long format.

EnumerationLiteral **ENUMErrorFormat::eFormatText**

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

7.3 RaptorXMLDev_COM - ENUMLoadSchemalocation

Enumeration **ENUMLoadSchemalocation**

diagram	
typedElements	Interface **IXBRL** — Operation **SchemalocationHints** Interface **IXMLValidator** — Operation **SchemalocationHints** Interface **IXSLT** — 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**

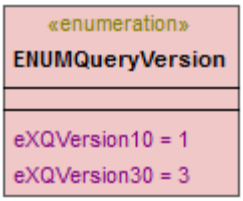
documentation	Sets the Load Schema Location to CombiningBoth. If either the namespace or URL has a catalog mapping, then the catalog mapping is used. If both have catalog mappings, then the value of the schema-mapping parameter decides which mapping is used. If neither the namespace nor URL has a catalog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

7.4 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**

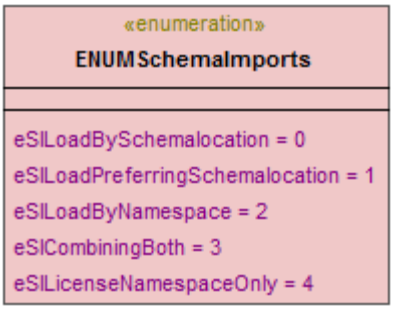
documentation	Sets the XQuery version you are going to use to XQuery 1.0.

EnumerationLiteral **ENUMQueryVersion::eXQVersion30**

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

7.5 RaptorXMLDev_COM - ENUMSchemaImports

Enumeration **ENUMSchemaImports**

diagram	
typedElements	Interface **IXBRL** Operation **SchemaImports** Interface **IXMLValidator** Operation **SchemaImports** Interface **IXSLT** 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**

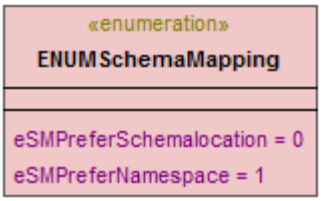
documentation	Sets the Schema Import to LoadBySchemalocation. The value of the schemaLocation attribute is used to locate the schema, taking account of catalog mappings. If the namespace attribute is present, the namespace is imported (licensed).

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.

7.6 RaptorXMLDev_COM - ENUMSchemaMapping

Enumeration **ENUMSchemaMapping**

diagram	
typedElements	Interface **IXBRL** Operation **SchemaMapping** Interface **IXMLValidator** Operation **SchemaMapping** Interface **IXSLT** 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.

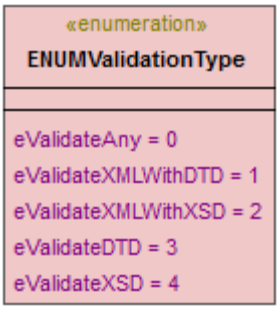
EnumerationLiteral **ENUMSchemaMapping::eSMPreferSchemalocation**

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

7.7 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**

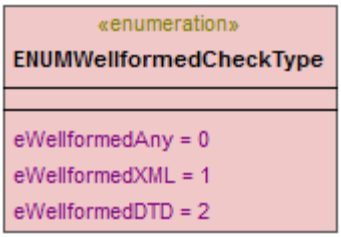
documentation	Sets the validation type to 'XML with XSD'. This validates an XML document against an XML Schema.

EnumerationLiteral **ENUMValidationType::eValidateXSD**

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

7.8 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**

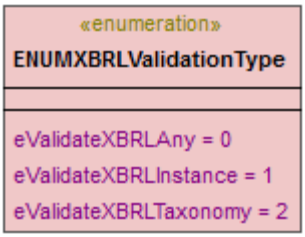
documentation	Sets the well-formed check type to 'DTD' This checks one or more DTD documents for well-formedness according to the XML 1.0 or XML 1.1 specification. Multiple documents can only be checked by RaptorXML Server, or RaptorXML+XBRL Server.

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.

7.9 RaptorXMLDev_COM - ENUMXBRLValidationType

Enumeration **ENUMXBRLValidationType**

diagram	
typedElements	Interface **IXBRL** Operation **IsValid**
documentation	Contains enumeration literals defining the type of XBRL document that will be validated. Only available for RaptorXML+XBRL

EnumerationLiteral **ENUMXBRLValidationType::eValidateXBRLAny**

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

EnumerationLiteral **ENUMXBRLValidationType::eValidateXBRLInstance**

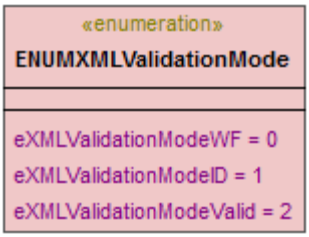
documentation	Sets the validation type to 'instance'. This validates the XBRL instance document. The document type is detected automatically.

EnumerationLiteral **ENUMXBRLValidationType::eValidateXBRLTaxonomy**

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

7.10 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'.

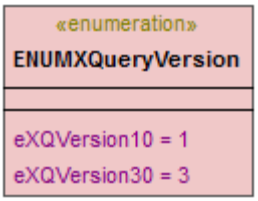
EnumerationLiteral **ENUMXMLValidationMode::eXMLValidationModeWF**

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

7.11 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.

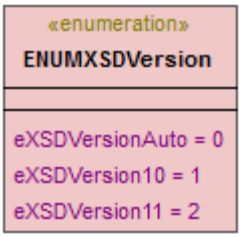
EnumerationLiteral **ENUMXQueryVersion::eXQVersion30**

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

7.12 RaptorXMLDev_COM - ENUMXSDVersion

Enumeration **ENUMXSDVersion**

diagram	
typedElements	Interface **IXMLValidator** Operation **XSDVersion** Interface **IXQuery** Operation **XSDVersion** Interface **IXSLT** 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.

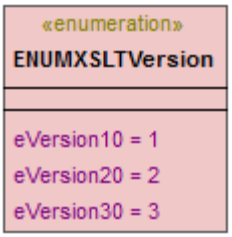
EnumerationLiteral **ENUMXSDVersion::eXSDVersionAuto**

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

7.13 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.

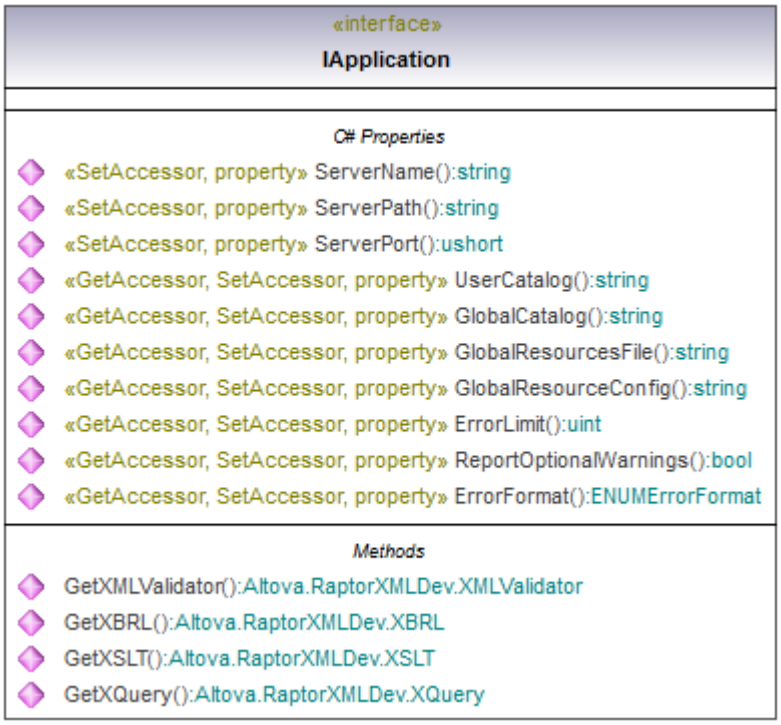
EnumerationLiteral **ENUMXSLTVersion::eVersion30**

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

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

7.14 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.					

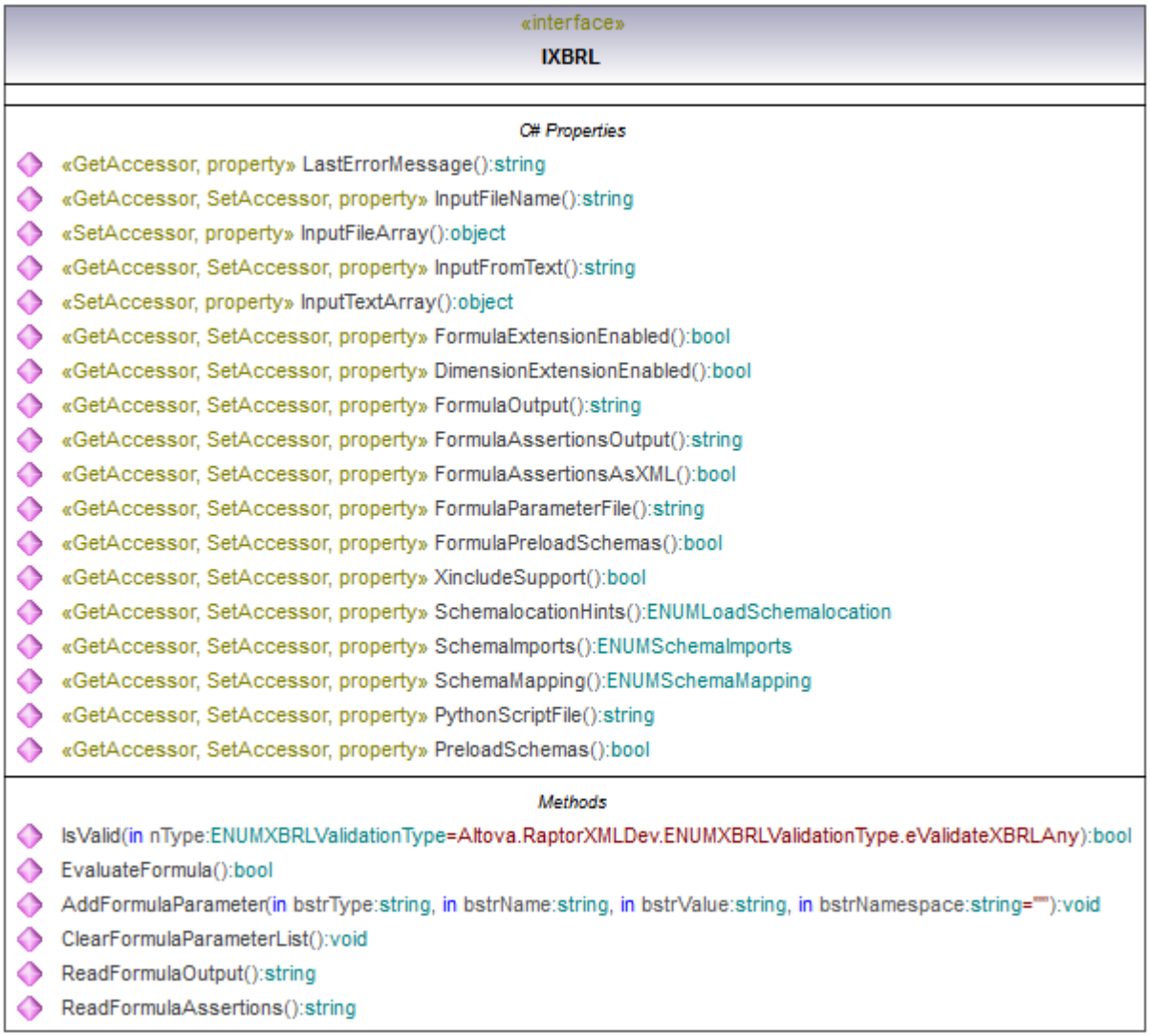
Operation **IApplication::UserCatalog**

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.					

7.15 RaptorXMLDev_COM - IXBRL

Interface **IXBRL**

diagram



documentation

Validates the supplied XML or XBRL (Note: XBRL only with RaptorXML+XBRL) document against internal or external DTDs or XML Schemas.

Checks well-formedness of XML, DTD, XML Schema, XSLT, and XQuery documents.

Validates XBRL taxonomies, and XBRL documents against XBRL taxonomies (only RaptorXML+XBRL).

Operation **IXBRL::AddFormulaParameter**

parameter	name	direction	type	type modifier	multiplicity	default
	bstrType	**in**	**string**			
	bstrName	**in**	**string**			
	bstrValue	**in**	**string**			
	bstrNamespace	**in**	**string**			**""**

	return	**return**	**void**
documentation	Adds a parameter used in the formula evaluation process. Properties: 'bstrtype' is the parameter data type - of type string. 'bstrname' is the parameter name - of type string. 'bstrvalue' is the parameter value - of type string. 'bstrnamespace' is the parameter namespace - of type string.		

Operation **IXBRL::ClearFormulaParameterList**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**void**			
documentation	Clears the list of formula parameters created with the AddFormulaParameter method. Properties: none.					

Operation **IXBRL::DimensionExtensionEnabled**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			
documentation	Enables XBRL Dimension extension validation. Properties: Returns a boolean value. 'true' enables the dimension extension validaton, 'false' disables it.					

Operation **IXBRL::EvaluateFormula**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			
documentation	Evaluates XBRL formulas in an XBRL instance file. Returns: 'true' on success, and false on false on failure. If an error occurs during execution, use the getLastErrorMessage operation to access additional information. Raises a RaptorXMLException when an error occurs.					

Operation **IXBRL::FormulaAssertionsAsXML**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			
documentation	Enables XML formatting of the assertion file when RaptorXML is run with assertions enabled. Properties: Returns a boolean value.					

	'true' enables XML output, 'false' generates JSON output.

Operation **IXBRL::FormulaAssertionsOutput**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			
documentation	Sets the location of the formula assertion output file. Properties: The full path of the output file - of type string.					

Operation **IXBRL::FormulaExtensionEnabled**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			
documentation	Enables XBRL formula extensions validation. Properties: Returns a boolean value. 'true' enables formula extension validation, 'false' disables it.					

Operation **IXBRL::FormulaOutput**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			
documentation	Sets the location for the output of the XBRL formula evaluation file. Properties: The full path of the formula output file - of type string.					

Operation **IXBRL::FormulaParameterFile**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			
documentation	Sets the location for the output of the formula parameter file. Properties: The full path of the formula parameter file - of type string.					

Operation **IXBRL::FormulaPreloadSchemas**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			
documentation	Defines if the XBRL 2.1 schemas will be preloaded. Properties: 'true' preloads schemas of the XBRL schemas. Default is 'false'.					

Operation **IXBRL::InputFileArray**

parameter	name	direction	type	type modifier	multiplicity	default

r	return	return	object
documentation	Sets the array of XBRL files that will be used as input data/instances. Properties: Object containing the strings of the absolute URLs of each of the input files.		

Operation **IXBRL::InputFileName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			
documentation	Sets the file name for the XBRL input instance file. Properties: A string containing the absolute URL (or base location) of the input file.					

Operation **IXBRL::InputFromText**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			
documentation	Supplies the contents of the XBRL input document as text. Properties: Contains the XBRL data as text - of type string.					

Operation **IXBRL::InputTextArray**

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

Operation **IXBRL::IsValid**

parameter	name	direction	type	type modifier	multiplicity	default
	nType	**in**	**ENUMXBRLValidationType**			**Altova.RaptorXMLDev.ENUMXBRLValidationType.eValidateXBRLAny**
	return	**return**	**bool**			
documentation	Result of the XBRL validation. Properties: 'nType' the value of the ENUMXBRLValidationType.					

	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 **IXBRL::LastErrorMessage**

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

Operation **IXBRL::PreloadSchemas**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**bool**			
documentation	Defines if the XBRL 2.1 schemas will be preloaded. Values: 'true' preloads schemas of the XBRL schemas. Default is 'false'.					

Operation **IXBRL::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. Parameters: An absolute URL that gives the base location of the file - of type string.					

Operation **IXBRL::ReadFormulaAssertions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			
documentation	Reads the formula assertion evaluation from the specified file. Parameters: none.					

Operation **IXBRL::ReadFormulaOutput**

parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**string**			
documentation	Reads the output of the formula assertion evaluation from the specified file. Parameters:					

	none.

Operation **IXBRL::SchemaImports**

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

Operation **IXBRL::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 **IXBRL::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.					

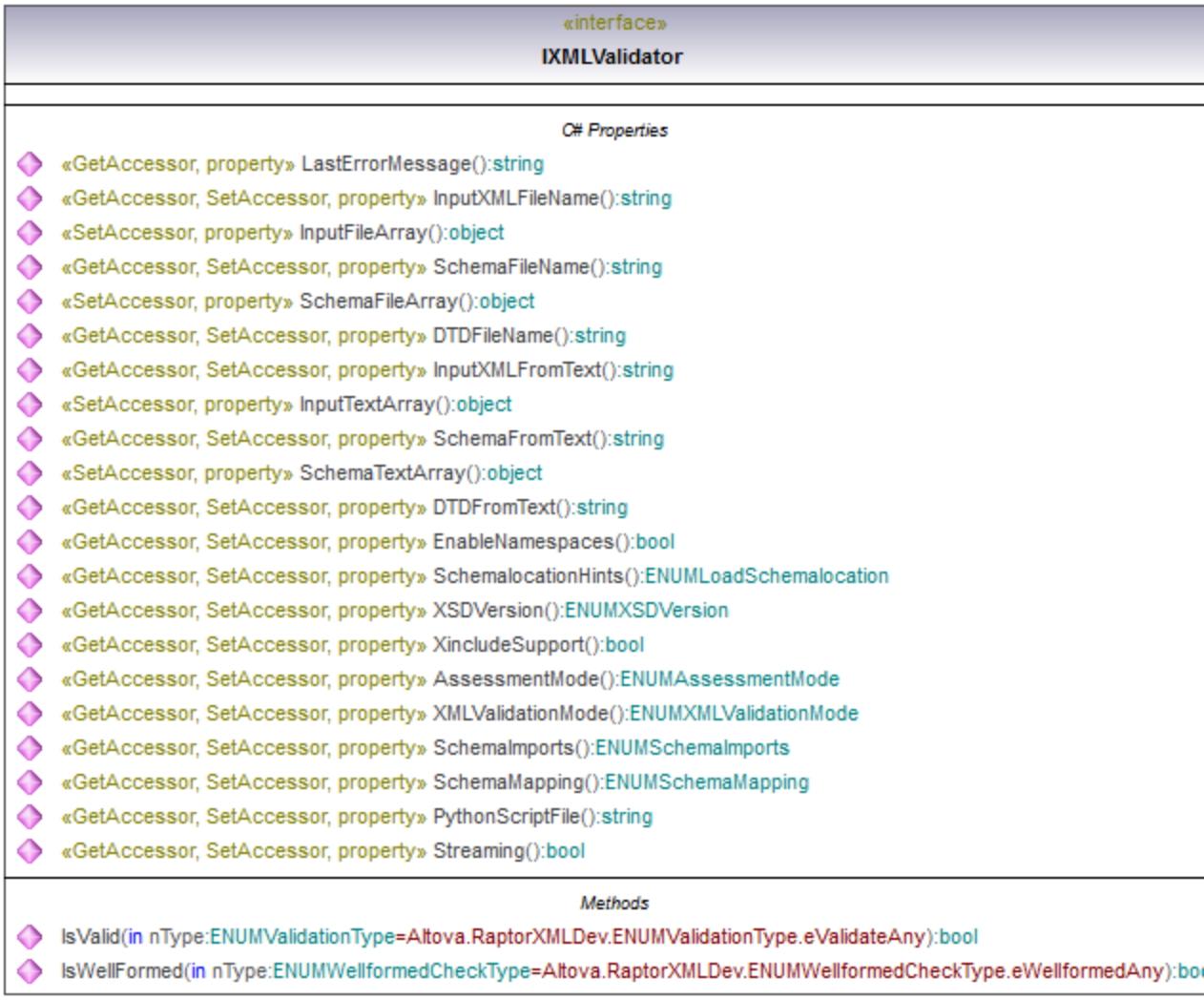
Operation **IXBRL::XincludeSupport**

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

7.16 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.					

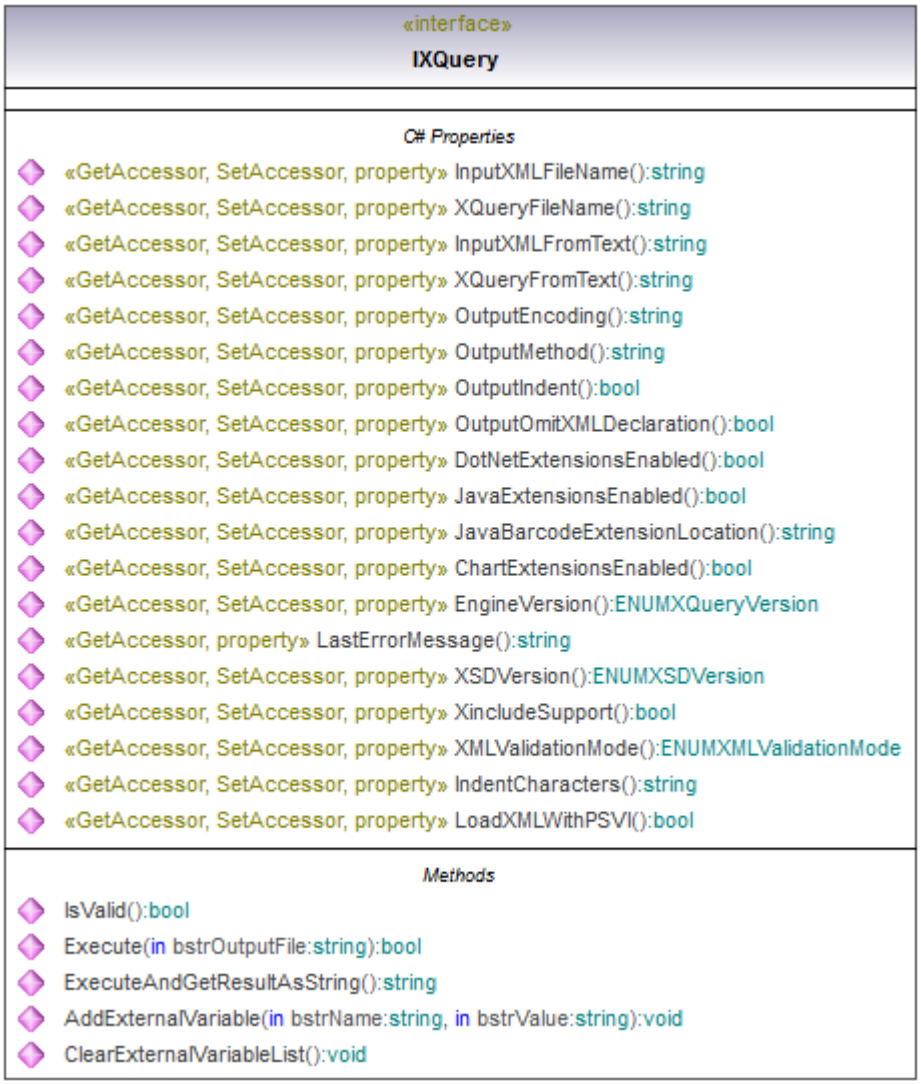
Operation **IXMLValidator::XSDVersion**

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.					

7.17 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**

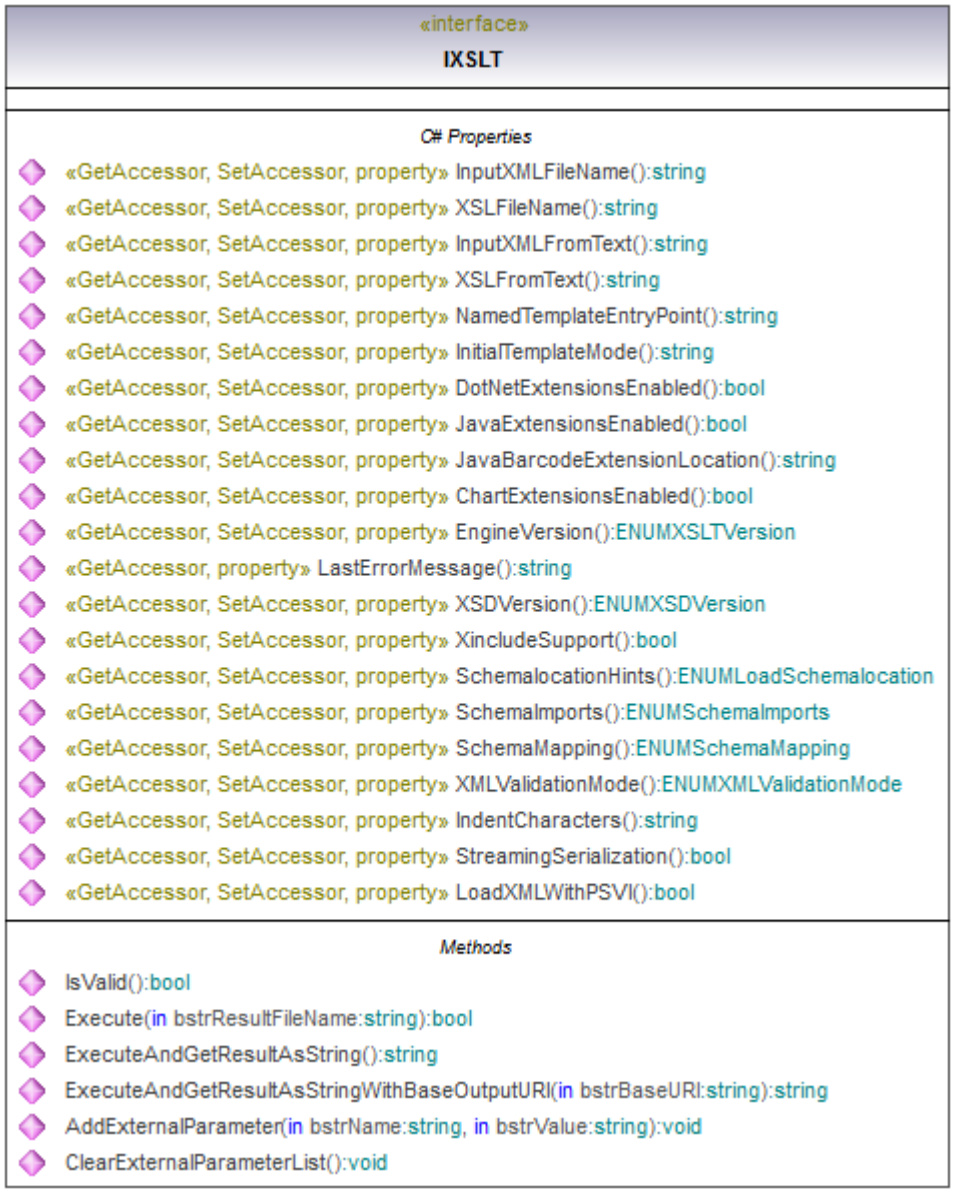
parameter	name	direction	type	type modifier	multiplicity	default
	return	**return**	**ENUMXSDVersion**			
documentation	Defines the XML Schema version that the document will be validated against. Returns the XML Schema version set by the EnumXSDVersion literal - of type EnumXSDVersion.					

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

7.18 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.					

Chapter 8

XSLT and XQuery Engine Information

8 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:

8.1 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.

8.2 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.

8.3 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.

8.4 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.

8.5 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.

8.6 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.

8.7 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 9

XSLT and XQuery Extension Functions

9 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 XBRL functions for XSLT, MSXSL scripts for XSLT and Altova's own extension functions.

You should note that, with the exception of some Altova extension functions for XSLT and the XBRL 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
- XBRL functions for XSLT
- 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.

9.1 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()

Chart functions (Enterprise and Server Editions only)
Altova extension functions for charts are supported only in the Enterprise and Server Editions of Altova products and enable charts to be generated from XML data. The chart functions have been organized into two groups:

- Functions to generate and save charts
- Functions to create charts

A third section gives a listing of the chart data XML structure, from which charts can be generated. Finally, an example XSLT document shows how chart functions can be used to generate charts from XML data.

9.1.1 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()*
```

9.1.2 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>
```

9.1.3 Chart Functions

The chart functions listed below enable you to create, generate, and save charts as images. They are supported in the current version of your Altova product in the manner described below. However, note that in future versions of your product, support for one or more of these functions might be discontinued or the behavior of individual functions might change. Consult the documentation of future releases for information about support for Altova extension functions in that release.

The chart functions are XPath functions (not XSLT functions), and organized into two groups:

- Functions for generating and saving charts
- Functions for creating charts

Note: Chart functions are supported only in **Altova's Server products** and the **Enterprise Editions of Altova products**.

Note: Supported image formats for charts in server editions are `jpg`, `png`, and `bmp`. The best option is png because it is lossless and compressed. In Enterprise editions, the supported formats are `jpg`. `png`, `bmp`, and `gif`.

Functions for generating and saving charts
These functions take the chart object (obtained with the chart creation functions) and either generate an image or save an image to file

```
altova:generate-chart-image ($chart, $width, $height, $encoding) as atomic
```

where

- `$chart` is the chart extension item obtained with the `altova:create-chart` function
- `$width` and `$height` must be specified with a length unit
- `$encoding` may be `binarytobase64` or `binarytobase16`

The function returns the chart image in the specified encoding.

```
altova:generate-chart-image ($chart, $width, $height, $encoding, $imagetype)
as atomic
```

where

- `$chart` is the chart extension item obtained with the `altova:create-chart` function
- `$width` and `$height` must be specified with a length unit
- `$encoding` may be `base64Binary` or `hexBinary`
- `$imagetype` may be one of the following image formats: `png`, `gif`, `bmp`, `jpg`, `jpeg`. Note that `gif` is not supported on server products. *Also see note at top of page*.

The function returns the chart image in the specified encoding and image format.

---

altova:save-chart-image ($chart, $filename, $width, $height) as empty()
(Windows only)

where

- $chart is the chart extension item obtained with the altova:create-chart function
- $filename is the path to and name of the file to which the chart image is to be saved
- $width and $height must be specified with a length unit

The function saves the chart image to the file specified in $filename.

---

altova:save-chart-image ($chart, $filename, $width, $height, $imagetype) as empty() ***(Windows only)***

where

- $chart is the chart extension item obtained with the altova:create-chart function
- $filename is the path to and name of the file to which the chart image is to be saved
- $width and $height must be specified with a length unit
- $imagetype may be one of the following image formats: png, gif, bmp, jpg, jpeg. Note that gif is not supported on server products. *Also see note at top of page*.

The function saves the chart image to the file specified in $filename in the image format specified.

Functions for creating charts
The following functions are used to create charts.

altova:create-chart($chart-config, $chart-data-series*) as chart extension item

where

- $chart-config is the chart-config extension item obtained with the altova:create-chart-config function or or via the altova:create-chart-config-from-xml function
- $chart-data-series is the chart-data-series extension item obtained with the altova:create-chart-data-series function or altova:create-chart-data-series-from-rows function

The function returns a chart extension item, which is created from the data supplied via the arguments.

---

altova:create-chart-config($type-name, $title) as chart-config extension item

where

- `$type-name` specifies the type of chart to be created: `Pie, Pie3d, BarChart, BarChart3d, BarChart3dGrouped, LineChart, ValueLineChart, RoundGauge, BarGauge`
- `$title` is the name of the chart

The function returns a chart-config extension item containing the configuration information of the chart.

---

altova:create-chart-config-from-xml`($xml-struct) as chart-config extension item`

where

- `$xml-struct` is the XML structure containing the configuration information of the chart

The function returns a chart-config extension item containing the configuration information of the chart. This information is supplied in an XML data fragment.

---

altova:create-chart-data-series`($series-name?, $x-values*, $y-values*) as chart-data-series extension item`

where

- `$series-name` specifies the name of the series
- `$x-values` gives the list of X-Axis values
- `$y-values` gives the list of Y-Axis values

The function returns a chart-data-series extension item containing the data for building the chart: that is, the names of the series and the Axes data.

---

altova:create-chart-data-row`(x, y1, y2, y3, ...) as chart-data-x-Ny-row extension item`

where

- `x` is the value of the X-Axis column of the chart data row
- `yN` are the values of the Y-Axis columns

The function returns a chart-data-x-Ny-row extension item, which contains the data for the X-Axis column and Y-Axis columns of a single series.

---

altova:create-chart-data-series-from-rows($series-names as xs:string*, $row*) as chart-data-series extension item

where

- $series-name is the name of the series to be created
- $row is the chart-data-x-Ny-row extension item that is to be created as a series

The function returns a chart-data-series extension item, which contains the data for the X-Axis and Y-Axes of the series.

---

altova:create-chart-layer($chart-config, $chart-data-series*) as chart-layer extension item

where

- $chart-config is the chart-config extension item obtained with the altova:create-chart-config function or or via the altova:create-chart-config-from-xml function
- $chart-data-series is the chart-data-series extension item obtained with the altova:create-chart-data-series function or altova:create-chart-data-series-from-rows function

The function returns a chart-layer extension item, which contains chart-layer data.

---

altova:create-multi-layer-chart($chart-config, $chart-data-series*, $chart-layer*)

where

- $chart-config is the chart-config extension item obtained with the altova:create-chart-config function or or via the altova:create-chart-config-from-xml function
- $chart-data-series is the chart-data-series extension item obtained with the altova:create-chart-data-series function or altova:create-chart-data-series-from-rows function
- $chart-layer is the chart-layer extension item obtained with the altova:create-chart-layer function

The function returns a multi-layer-chart item.

---

altova:create-multi-layer-chart($chart-config, $chart-data-series*, $chart-layer*, xs:boolean $mergecategoryvalues)

where

- $chart-config is the chart-config extension item obtained with the

`altova:create-chart-config` function or or via the `altova:create-chart-config-from-xml` function
- `$chart-data-series` is the chart-data-series extension item obtained with the `altova:create-chart-data-series` function or `altova:create-chart-data-series-from-rows` function
- `$chart-layer` is the chart-layer extension item obtained with the `altova:create-chart-layer` function

The function returns a multi-layer-chart item.

9.1.4 Chart Data XML Structure

Given below is the XML structure of chart data, how it might appear for the Altova extension functions for charts. This affects the appearance of the specific chart. Not all elements are used for all chart kinds, e.g. the `<Pie>` element is ignored for bar charts.

Note: Chart functions are supported only in the **Enterprise and Server Editions** of Altova products.

```
<chart-config>
   <General
      SettingsVersion="1"   must be provided
      ChartKind="BarChart"   Pie, Pie3d, BarChart, StackedBarChart, BarChart3d,
BarChart3dGrouped, LineChart, ValueLineChart, AreaChart, StackedAreaChart, RoundGauge,
BarGauge, CandleStick
      BKColor="#ffffff"   Color
      BKColorGradientEnd="#ffffff"   Color. In case of a gradient, BKColor and
BKColorGradientEnd define the gradient's colors
      BKMode="#ffffff"   Solid, HorzGradient, VertGradient
      BKFile="Path+Filename"   String. If file exists, its content is drawn over the
background.
      BKFileMode="Stretch"   Stretch, ZoomToFit, Center, Tile
      ShowBorder="1"   Bool
      PlotBorderColor="#000000"   Color
      PlotBKColor="#ffffff"   Color
      Title=""   String
      ShowLegend="1"   Bool
      OutsideMargin="3.%"   PercentOrPixel
      TitleToPlotMargin="3.%"   PercentOrPixel
      LegendToPlotMargin="3.%"   PercentOrPixel
      Orientation="vert"   Enumeration: possible values are: vert, horz
      >
      <TitleFont
         Color="#000000"   Color
         Name="Tahoma"   String
         Bold="1"   Bool
         Italic="0"   Bool
         Underline="0"   Bool
         MinFontHeight="10.pt"   FontSize (only pt values)
         Size="8.%"   FontSize />
      <LegendFont
         Color="#000000"
         Name="Tahoma"
         Bold="0"
         Italic="0"
         Underline="0"
         MinFontHeight="10.pt"
         Size="3.5%" />
      <AxisLabelFont
         Color="#000000"
         Name="Tahoma"
         Bold="1"
         Italic="0"
         Underline="0"
         MinFontHeight="10.pt"
         Size="5.%" />
   </General>
```

```
<Line
    ConnectionShapeSize="1.%"   PercentOrPixel
    DrawFilledConnectionShapes="1"   Bool
    DrawOutlineConnectionShapes="0"   Bool
    DrawSlashConnectionShapes="0"   Bool
    DrawBackslashConnectionShapes="0"   Bool
/>

<Bar
    ShowShadow="1"   Bool
    ShadowColor="#a0a0a0"   Color
    OutlineColor="#000000"   Color
    ShowOutline="1"   Bool
/>

<Area
    Transparency="0"   UINT ( 0-255 ) 255 is fully transparent, 0 is opaque
    OutlineColor="#000000"   Color
    ShowOutline="1"   Bool
/>

<CandleStick
    FillHighClose="0"   Bool. If 0, the body is left empty. If 1, FillColorHighClose is used
for the candle body
    FillColorHighClose="#ffffff"   Color. For the candle body when close > open
    FillHighOpenWithSeriesColor="1"   Bool. If true, the series color is used to fill the
candlebody when open > close
    FillColorHighOpen="#000000"   Color. For the candle body when open > close and
FillHighOpenWithSeriesColor is false
/>

<Colors   User-defined color scheme: By default this element is empty except for the style
and has no Color attributes
    UseSubsequentColors ="1"   Boolean. If 0, then color in overlay is used. If 1, then
subsequent colors from previous chart layer is used
    Style="User"   Possible values are: "Default", "Grayscale", "Colorful", "Pastel", "User"
    Colors="#52aca0"   Color: only added for user defined color set
    Colors1="#d3c15d"   Color: only added for user defined color set
    Colors2="#8971d8"   Color: only added for user defined color set
    ...
    ColorsN=""   Up to ten colors are allowed in a set: from Colors to Colors9
</Colors>

<Pie
    ShowLabels="1"   Bool
    OutlineColor="#404040"   Color
    ShowOutline="1"   Bool
    StartAngle="0."   Double
    Clockwise="1"   Bool
    Draw2dHighlights="1"   Bool
    Transparency="0"   Int (0 to 255: 0 is opaque, 255 is fully transparent)
    DropShadowColor="#c0c0c0"   Color
    DropShadowSize="5.%"   PercentOrPixel
    PieHeight="10.%"   PercentOrPixel. Pixel values might be different in the result
because of 3d tilting
    Tilt="40.0"   Double (10 to 90: The 3d tilt in degrees of a 3d pie)
    ShowDropShadow="1"   Bool
    ChartToLabelMargin="10.%"   PercentOrPixel
```



```
        AddValueToLabel="0"   Bool
        AddPercentToLabel="0"   Bool
        AddPercentToLabels_DecimalDigits="0"  UINT ( 0 – 2 )
        >
        <LabelFont
            Color="#000000"
            Name="Arial"
            Bold="0"
            Italic="0"
            Underline="0"
            MinFontHeight="10.pt"
            Size="4.%"    />
    </Pie>

    <XY>
        <XAxis   Axis
            AutoRange="1"   Bool
            AutoRangeIncludesZero="1"   Bool
            RangeFrom="0."   Double: manual range
            RangeTill="1."   Double : manual range
            LabelToAxisMargin="3.%"   PercentOrPixel
            AxisLabel=""   String
            AxisColor="#000000"   Color
            AxisGridColor="#e6e6e6"   Color
            ShowGrid="1"   Bool
            UseAutoTick="1"   Bool
            ManualTickInterval="1."   Double
            AxisToChartMargin="0.px"   PercentOrPixel
            TickSize="3.px"   PercentOrPixel
            ShowTicks="1"   Bool
            ShowValues="1"   Bool
            AxisPosition="LeftOrBottom"   Enums: "LeftOrBottom", "RightOrTop",
"AtValue"
            AxisPositionAtValue = "0"   Double
            >
            <ValueFont
                Color="#000000"
                Name="Tahoma"
                Bold="0"
                Italic="0"
                Underline="0"
                MinFontHeight="10.pt"
                Size="3.%"    />
        </XAxis>
        <YAxis   Axis (same as for XAxis)
            AutoRange="1"
            AutoRangeIncludesZero="1"
            RangeFrom="0."
            RangeTill="1."
            LabelToAxisMargin="3.%"
            AxisLabel=""
            AxisColor="#000000"
            AxisGridColor="#e6e6e6"
            ShowGrid="1"
            UseAutoTick="1"
            ManualTickInterval="1."
            AxisToChartMargin="0.px"
            TickSize="3.px"
            ShowTicks="1"   Bool
            ShowValues="1"   Bool
```

```
            AxisPosition="LeftOrBottom"   Enums: "LeftOrBottom", "RightOrTop",
"AtValue"
            AxisPositionAtValue = "0"   Double
            >
            <ValueFont
               Color="#000000"
               Name="Tahoma"
               Bold="0"
               Italic="0"
               Underline="0"
               MinFontHeight="10.pt"
               Size="3.%"/>
         </YAxis>
      </XY>

      <XY3d
         AxisAutoSize="1"   Bool: If false, XSize and YSize define the aspect ration of x and y
axis. If true, aspect ratio is equal to chart window
         XSize="100.%"   PercentOrPixel. Pixel values might be different in the result because
of 3d tilting and zooming to fit chart
         YSize="100.%"   PercentOrPixel. Pixel values might be different in the result because
of 3d tilting and zooming to fit chart
         SeriesMargin="30.%"   PercentOrPixel. Pixel values might be different in the result
because of 3d tilting and zooming to fit chart
         Tilt="20."   Double. -90 to +90 degrees
         Rot="20."   Double. -359 to +359 degrees
         FoV="50.">   Double. Field of view: 1-120 degree
         >
         <ZAxis
            AutoRange="1"
            AutoRangeIncludesZero="1"
            RangeFrom="0."
            RangeTill="1."
            LabelToAxisMargin="3.%"
            AxisLabel=""
            AxisColor="#000000"
            AxisGridColor="#e6e6e6"
            ShowGrid="1"
            UseAutoTick="1"
            ManualTickInterval="1."
            AxisToChartMargin="0.px"
            TickSize="3.px" >
            <ValueFont
               Color="#000000"
               Name="Tahoma"
               Bold="0"
               Italic="0"
               Underline="0"
               MinFontHeight="10.pt"
               Size="3.%"/>
         </ZAxis>
      </XY3d>

      <Gauge
         MinVal="0."   Double
         MaxVal="100."   Double
         MinAngle="225"   UINT: -359-359
         SweepAngle="270"   UINT: 1-359
         BorderToTick="1.%"   PercentOrPixel
         MajorTickWidth="3.px"   PercentOrPixel
         MajorTickLength="4.%"   PercentOrPixel
         MinorTickWidth="1.px"   PercentOrPixel
```



```
        MinorTickLength="3.%"   PercentOrPixel
        BorderColor="#a0a0a0"   Color
        FillColor="#303535"   Color
        MajorTickColor="#a0c0b0"   Color
        MinorTickColor="#a0c0b0"   Color
        BorderWidth="2.%"   PercentOrPixel
        NeedleBaseWidth="1.5%"   PercentOrPixel
        NeedleBaseRadius="5.%"   PercentOrPixel
        NeedleColor="#f00000"   Color
        NeedleBaseColor="#141414"   Color
        TickToTickValueMargin="5.%"   PercentOrPixel
        MajorTickStep="10."   Double
        MinorTickStep="5."   Double
        RoundGaugeBorderToColorRange="0.%"   PercentOrPixel
        RoundGaugeColorRangeWidth ="6.%"   PercentOrPixel
        BarGaugeRadius="5.%"   PercentOrPixel
        BarGaugeMaxHeight="20.%"   PercentOrPixel
        RoundGaugeNeedleLength="45.%"   PercentOrPixel
        BarGaugeNeedleLength="3.%"   PercentOrPixel
        >
        <TicksFont
            Color="#a0c0b0"
            Name="Tahoma"
            Bold="0"
            Italic="0"
            Underline="0"
            MinFontHeight="10.pt"
            Size="4.%"
        />
        <ColorRanges>   User-defined color ranges. By default empty with no child element
entries
            <Entry
                From="50. "   Double
                FillWithColor="1"   Bool
                Color="#00ff00"   Color
            />
            <Entry
                From="50.0"
                FillWithColor="1"
                Color="#ff0000"
            />
            ...
        </ColorRanges>
    </Gauge>
</chart-config>
```

9.1.5 Example: Chart Functions

The example XSLT document below shows how Altova extension functions for charts can be used. Given further below are an XML document and a screenshot of the output image generated when the XML document is processed with the XSLT document using the XSLT 2.0 or 3.0 Engine.

Note: Chart functions are supported only in the **Enterprise and Server Editions** of Altova products.

Note: For more information about how chart data tables are created, see the documentation of Altova's XMLSpy and StyleVision products.

XSLT document
This XSLT document (*listing below*) uses Altova chart extension functions to generate a pie chart. It can be used to process the XML document listed further below.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:altovaext="http://www.altova.com/xslt-extensions"
    exclude-result-prefixes="#all">
    <xsl:output version="4.0" method="html" indent="yes" encoding="UTF-8"/>
    <xsl:template match="/">
        <html>
            <head>
                <title>
                    <xsl:text>HTML Page with Embedded Chart</xsl:text>
                </title>
            </head>
            <body>
                <xsl:for-each select="/Data/Region[1]">
                    <xsl:variable name="extChartConfig" as="item()*">
                        <xsl:variable name="ext-chart-settings" as="item()*">
                            <chart-config>
                                <General
                                    SettingsVersion="1"
                                    ChartKind="Pie3d"
                                    BKColor="#ffffff"
                                    ShowBorder="1"
                                    PlotBorderColor="#000000"
                                    PlotBKColor="#ffffff"
                                    Title="{@id}"
                                    ShowLegend="1"
                                    OutsideMargin="3.2%"
                                    TitleToPlotMargin="3.%"
                                    LegendToPlotMargin="6.%"
                                    >
                                    <TitleFont
                                        Color="#023d7d"
                                        Name="Tahoma"
                                        Bold="1"
                                        Italic="0"
                                        Underline="0"
                                        MinFontHeight="10.pt"
                                        Size="8.%" />
                                </General>
                            </chart-config>
                        </xsl:variable>
                        <xsl:sequence select="
```



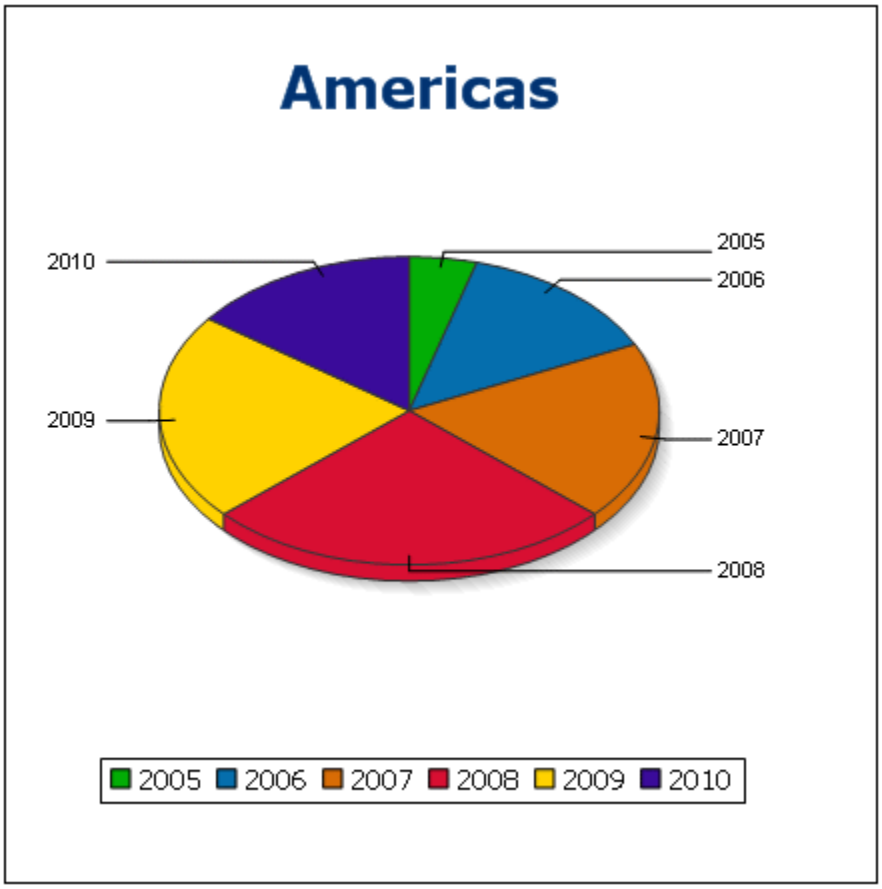
```
altovaext:create-chart-config-from-xml( $ext-chart-settings )"/>
                </xsl:variable>
                <xsl:variable name="chartDataSeries" as="item()*">
                   <xsl:variable name="chartDataRows" as="item()*">
                      <xsl:for-each select="(Year)">
                         <xsl:sequence select="
altovaext:create-chart-data-row( (@id), ( .) )"/>
                      </xsl:for-each>
                   </xsl:variable>
                   <xsl:variable name="chartDataSeriesNames" as="xs:string*"
select=" ( (&quot;Series 1&quot;), &apos;&apos; )[1]"/>
                   <xsl:sequence
                      select="altovaext:create-chart-data-series-from-rows(
$chartDataSeriesNames, $chartDataRows)"/>
                </xsl:variable>
                <xsl:variable name="ChartObj" select="altovaext:create-chart(
$extChartConfig, ( $chartDataSeries), false() )"/>
                <xsl:variable name="sChartFileName" select="'mychart1.png'"/>
                <img src="{$sChartFileName, altovaext:save-chart-image(
$ChartObj, $sChartFileName, 400, 400 ) }"/>
             </xsl:for-each>
          </body>
       </html>
   </xsl:template>
</xsl:stylesheet>
```

XML document

This XML document can be processed with the XSLT document above. Data in the XML document is used to generate the pie chart shown in the screenshot below.

```
<?xml version="1.0" encoding="UTF-8"?>
<Data xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="YearlySales.xsd">
   <ChartType>Pie Chart 2D</ChartType>
   <Region id="Americas">
      <Year id="2005">30000</Year>
      <Year id="2006">90000</Year>
      <Year id="2007">120000</Year>
      <Year id="2008">180000</Year>
      <Year id="2009">140000</Year>
      <Year id="2010">100000</Year>
   </Region>
   <Region id="Europe">
      <Year id="2005">50000</Year>
      <Year id="2006">60000</Year>
      <Year id="2007">80000</Year>
      <Year id="2008">100000</Year>
      <Year id="2009">95000</Year>
      <Year id="2010">80000</Year>
   </Region>
   <Region id="Asia">
      <Year id="2005">10000</Year>
      <Year id="2006">25000</Year>
      <Year id="2007">70000</Year>
      <Year id="2008">110000</Year>
      <Year id="2009">125000</Year>
      <Year id="2010">150000</Year>
   </Region>
</Data>
```

Output image
The pie chart show below is generated when the XML document listed above is processed with the XSLT document.



9.2 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.

9.2.1 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.

9.2.2 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.

9.2.3 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" />
```

9.2.4 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>
```

9.2.5 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`.

9.2.6 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.

9.2.7 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).

9.3 .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.

9.3.1 .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" />
```
```

9.3.2 .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>
```


9.3.3 .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.

9.3.4 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.

9.3.5 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).

9.4 XBRL Functions for XSLT

Functions defined in the XBRL function registry can be called from within an XSLT context for transforming XBRL instance documents. These XBRL functions are defined in one of two namespaces:

`http://www.xbrl.org/2008/function/instance` (usually used with the `xfi:` prefix)
`http://www.xbrl.org/2010/function/formula` (usually used with the `xff:` prefix)

So the XBRL function `xfi:context`, for example, expands to `http://www.xbrl.org/2008/function/instance:context` (assuming this namespace has been bound to the `xfi:` prefix).

For a complete list of the functions, go to http://www.xbrl.org/functionregistry/functionregistry.xml.

9.5 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.

Chapter 10

Altova LicenseServer

10 Altova LicenseServer

Altova LicenseServer (hereafter also called LicenseServer) provides a central location for the management of licenses for Altova products. Altova applications running in a network can have licenses assigned to them from the LicenseServer, thus giving administrators the flexibility to manage and monitor licenses.

Licensing process with Altova LicenseServer
To assign an Altova server product a license using Altova LicenseServer, you need to do the following:

1. Start LicenseServer.
2. Open the LicenseServer Configuration page, which is the administrator's interface with LicenseServer, on Windows or Linux.
3. Upload the license/s you have received from Altova to the license pool of your Altova LicenseServer. Do this in the License Pool tab of the LicenseServer Configuration page.
4. Register the Altova server product (FlowForce Server, MapForce Server, StyleVision Server, Register RaptorXML(+XBRL) Server with LicenseServer) with LicenseServer. Depending on the product's type, the method of registering it with LicenseServer will be different: either via the product's GUI or its command line. See the documentation of your Altova server product for information about how to register it with LicenseServer.
5. In the Server Management tab of the LicenseServer Configuration page, assign a license to the Altova server product according to the number of cores on the product machine.

Licenses can thereafter be conveniently monitored and managed centrally with LicenseServer. See the Configuration Page Reference for available functionality.

About this documentation
This documentation is organized into the following parts:

- Introductory information about: network requirements; installation on Windows and Linux; and Altova ServiceController.
- How to Assign Licenses, which describes in a step-by-step way how to assign licenses with Altova LicenseServer.
- Configuration Page Reference: A description of the administrator's interface with LicenseServer.

Note:
The LicenseServer administration interface does not support SSL.

10.1 Network Information

Altova LicenseServer must be installed on a server machine that is accessible by all clients running Altova products that require a license. Any firewall on both the client and server must allow the network traffic to and from the LicenseServer that is necessary for the LicenseServer to operate correctly.

On the LicenseServer, **port 35355** is used to distribute licenses, and therefore it must be open for network traffic with client machines.

The following are the default networking parameters and requirements of LicenseServer:

- *For LicenseServer license distribution:*
  Either one or both of
  IPv4 TCP connection on port `35355`
  IPv6 TCP connection on port `35355`

For administrative tasks, The LicenseServer is accessed by a web interface that uses port 8088. The port used can be configured to suit your requirements.

Connection to the Master Licensing Server at altova.com
The Altova LicenseServer needs to be able to communicate with the Master Licensing Server at `altova.com` to validate and authenticate license-related data and to ensure continuous compliance with the Altova license agreements. This communication occurs over HTTPS using port 443. If the Altova LicenseServer, after making the initial verification with the `altova.com` Master Licensing Server, is unable to again connect with `altova.com` for a duration of more than 5 days (= 120 hours), then the Altova LicenseServer will no longer permit the usage of any Altova software products connected to the Altova LicenseServer.

Any such loss of connection with the `altova.com` master servers will be logged in the Messages tab of the Configuration page of the Altova LicenseServer. In addition, the administrator can configure the Altova LicenseServer to automatically send an alert email when the connection to `altova.com` is lost. Alert Mail settings are available in the Settings tab of the Configuration page.

10.2 Installation (Windows)

Altova LicenseServer can be installed on Windows systems in one of two ways:

- As an independent installation.
- As part of an Altova server product installation. (Altova server products are: Altova FlowForce Server, Altova MapForce Server, and Altova SyleVision Server.)

If LicenseServer is not installed on your system at the time an Altova server product is installed, the option to install LicenseServer is selected by default during installation setup. If LicenseServer is already installed, the option to install it is deselected by default. You can change the default option if you like.

Note: If you wish to re-install Altova LicenseServer, you must first de-install the older version.

For information about how to proceed with assigning licenses, see the section How to Assign Licenses.

10.3 Installation (Linux)

Altova LicenseServer can be installed on Linux systems (Debian, Ubuntu, CentOS, RedHat).

Uninstalling old versions of LicenseServer

On the Linux command line interface (CLI), you can check whether LicenseServer is installed with the following command:

```
[Debian, Ubuntu]:   dpkg --list | grep Altova
[CentOS, RedHat]:   rpm -qa | grep server
```

If LicenseServer is not installed, go ahead with the installation as documented in the next steps. If LicenseServer is installed and you wish to install a newer version of it, uninstall the old version with the command:

```
[Debian, Ubuntu]:   sudo dpkg --remove licenseserver
[CentOS, RedHat]:   sudo rpm -e licenseserver
```

Installing Altova LicenseServer

On Linux systems, LicenseServer must be installed independently of other Altova server products. It is not included as part of the installation packages of Altova server products. Download Altova LicenseServer from the Altova website and copy the package to any directory on the Linux system.

Distribution	Installer extension
Debian	.deb
Ubuntu	.deb
CentOS	.rpm
RedHat	.rpm

In a terminal window, switch to the directory where you have copied the Linux package. For example, if you copied it to a user directory called MyAltova (that is located, say, in the /home/User directory), then switch to this directory as follows:

```
cd /home/User/MyAltova
```

Install LicenseServer with the following command:

```
[Debian]:   sudo dpkg --install licenseserver-1.1-debian.deb
[Ubuntu]:   sudo dpkg --install licenseserver-1.1-ubuntu.deb
[CentOS]:   sudo rpm -ivh licenseserver-1.1-1.x86_64.rpm
[RedHat]:   sudo rpm -ivh licenseserver-1.1-1.x86_64.rpm
```

The LicenseServer package will be installed in:

```
/opt/Altova/LicenseServer
```

For information about how to proceed with assigning licenses, see the section How to Assign

Licenses.

10.4 Installation (Mac OS X)

Altova LicenseServer can be installed on Mac OS X systems (version 10.7 or higher). Since you might need to uninstall a previous version, uninstalling is described first.

Uninstalling old versions of LicenseServer
Before uninstalling LicenseServer, stop the service with the following command:

```
sudo launchctl unload /Library/LaunchDaemons/com.altova.LicenserServer.plist
```

To check whether the service has been stopped, open the Activity Monitor terminal and make sure that LicenseServer is not in the list.

In the Applications terminal, right-click the LicenseServer icon and select **Move to Trash**. The application will be moved to Trash. You will, however, still need to remove the application from the `usr` folder. Do this with the command:

```
sudo rm -rf /usr/local/Altova/LicenseServer
```

Installing Altova LicenseServer
Download Altova LicenseServer from the Altova website (the installer file has a `.pkg` file extension), and double-click the installer package to start the installation. Follow the on-screen instructions. You will need to accept the license agreement for installation to proceed.

The LicenseServer package will be installed in the folder:

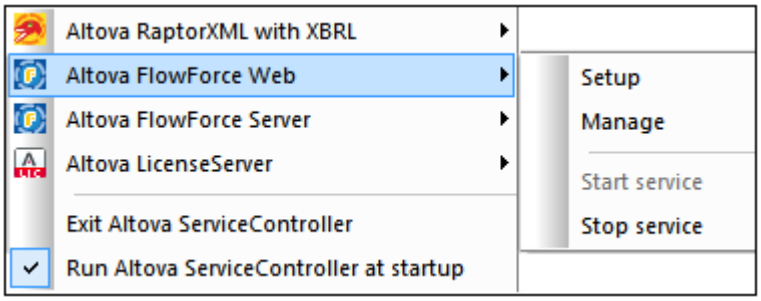
```
/usr/local/Altova/LicenseServer
```

10.5 Altova ServiceController

The Altova ServiceController (hereafter also called ServiceController) is an application for conveniently starting, stopping and configuring Altova services **on Windows systems**. It is not available on Linux systems.

Altova ServiceController is installed with Altova LicenseServer, and can be started by clicking its command in the Altova LicenseServer folder of the **Start** menu. After the ServiceController has been started, it can be accessed via the system tray (*screenshot below*).

To run the ServiceController after logging in to the system, click the ServiceController icon in the system tray to pop up the ServiceController menu (*screenshot below*), and then toggle on the command **Run Altova ServiceController at Startup**. (This command is toggled on by default.) To exit ServiceController, click the ServiceController icon in the system tray and, in the menu that pops up (*see screenshot below*), click **Exit Altova ServiceController**.



Starting and stopping Altova services
Each installed Altova service component will have an entry in the ServiceController menu (*see screenshot above*). An Altova service can be started or stopped via a command in its ServiceController sub-menu. Additionally, important administration tasks of individual services can be accessed via the ServiceController menu. In the screenshot above, for example, the Altova FlowForce Web service has a sub-menu in which you can choose to access its Setup page.

10.6 How to Assign Licenses

To assign an Altova server product a license using Altova LicenseServer, you need to do the following:

1. Start LicenseServer.
2. Open the LicenseServer Configuration page, which is the administrator's interface with LicenseServer, on Windows or Linux.
3. Upload the license/s you have received from Altova to the license pool of your Altova LicenseServer. Do this in the License Pool tab of the LicenseServer Configuration page.
4. Register the Altova server product (FlowForce Server, MapForce Server, StyleVision Server) with LicenseServer. Depending on the product's type, the method of registering it with LicenseServer will be different: either via the product's GUI or its command line. See the documentation of your Altova server product for information about how to register it with LicenseServer.
5. In the Server Management tab of the LicenseServer Configuration page, assign a license to the Altova server product according to the number of cores on the product machine.

Note on cores and licenses
The licensing of Altova server products is based on the number of processor cores available on the product machine. For example, a dual-core processor has two cores, a quad-core processor four cores, a hexa-core processor six cores, and so on. The number of cores licensed for a product on a particular server machine must be greater than or equal to the number of cores available on that server, whether it's a physical or virtual machine.

For example, if a server has eight cores (an octa-core processor), you must purchase at least an 8-core license. You can also combine licenses to achieve the core count. So, two 4-core licenses can also be used for an octa-core server instead of an 8-core license.

If you are using a computer server with a large number of CPU cores but only have a low volume to process, you may also create a virtual machine that is allocated a smaller number of cores, and purchase a license for that number. Such a deployment, of course, will have less processing speed than if all available cores on the server were utilized.

Note: Each license can be used for only one client machine at a time, even if it has unused licensing capacity. For example, if a 10-core license is used for a client machine that has 6 CPU cores, then the remaining 4 cores of the license cannot be used simultaneously for another client machine.

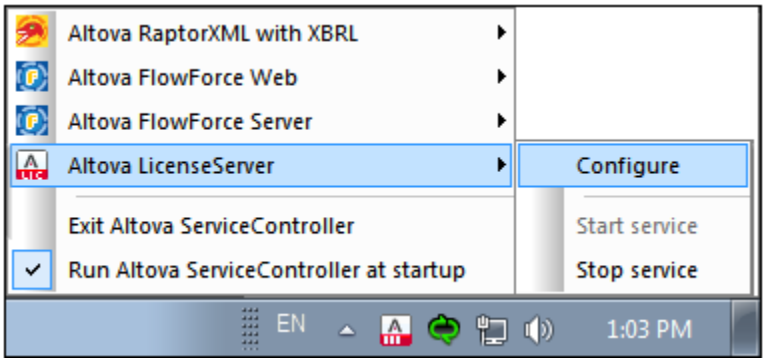
10.6.1 Start LicenseServer

This section:

- How to start LicenseServer on Windows systems
- How to start LicenseServer on Linux systems
- How to start LicenseServer on Mac OS X systems
- Note about Connection to altova.com

Windows systems
You can start LicenseServer via the Altova ServiceController, which is available in the system tray. (Click **Start | All Programs | Altova LicenseServer | Altova ServiceController** to start Altova ServiceController and display its icon in the system tray. If you select the *Run Altova ServiceController at Startup* option (*see screenshot below*), Altova ServiceController will start up on system start and its icon will be available in the system tray from then onwards.)

To start LicenseServer, click the Altova ServiceController icon in the system tray, mouse over **Altova LicenseServer** in the menu that pops up (*see screenshot below*), and then select **Start Service** from the LicenseServer submenu. If LicenseServer is already running, the *Start Service* option will be disabled.



Linux systems
To start LicenseServer as a service on Linux systems, run the following command in a terminal window.

```
[Debian]:          sudo /etc/init.d/licenseserver start

[Ubuntu]:          sudo initctl start licenseserver

[CentOS]:          sudo initctl start licenseserver

[RedHat]:          sudo initctl start licenseserver
```

(If you need to stop LicenseServer, replace `start` with `stop` in the above command)

Starting LicenseServer as a service
To start LicenseServer as a service on Mac OS X systems, run the following command in a

terminal window:

```
sudo launchctl load /Library/LaunchDaemons/com.altova.LicenserServer11.plist
```

If at any time you need to stop LicenseServer, use:

```
sudo launchctl unload /Library/LaunchDaemons/com.altova.LicenserServer11.
plist
```

Connection to the Master Licensing Server at altova.com
The Altova LicenseServer needs to be able to communicate with the Master Licensing Server at `altova.com` to validate and authenticate license-related data and to ensure continuous compliance with the Altova license agreements. This communication occurs over HTTPS using port 443. If the Altova LicenseServer, after making the initial verification with the `altova.com` Master Licensing Server, is unable to again connect with `altova.com` for a duration of more than 5 days (= 120 hours), then the Altova LicenseServer will no longer permit the usage of any Altova software products connected to the Altova LicenseServer.

Any such loss of connection with the `altova.com` master servers will be logged in the Messages tab of the Configuration page of the Altova LicenseServer. In addition, the administrator can configure the Altova LicenseServer to automatically send an alert email when the connection to `altova.com` is lost. Alert Mail settings are available in the Settings tab of the Configuration page.

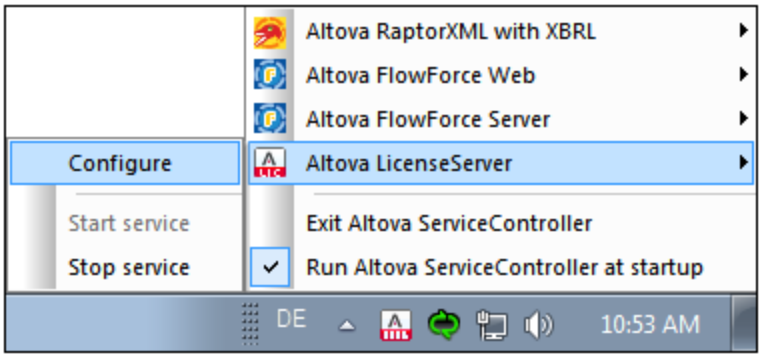
10.6.2 Open LicenseServer Config Page (Windows)

This section:

Opening the Configuration page if LicenseServer is on the same machine
On Windows systems, if LicenseServer is on the same machine, you can open the Configuration page of LicenseServer in one of two ways:

- Click **Start | All Programs | Altova LicenseServer | LicenseServer Configuration Page**. The Configuration page opens in a new tab of your Internet browser.

- Click the Altova ServiceController icon in the system tray, mouse over **Altova LicenseServer** in the menu that pops up (*see screenshot below*), and then select **Configure** from the LicenseServer submenu.



The Configuration page opens in a new browser window, and its login mask is displayed (*screenshot below*).

Opening the Configuration page if LicenseServer is on another machine
To open the LicenseServer Configuration page from some other Windows machine on the local network (than that on which LicenseServer is installed), enter the URL of the LicenseServer Configuration page in the address bar of a browser and press **Enter**. By default, the URL of the Configuration page will be:

```
http://<serverIPAddressOrName>:8088/
```

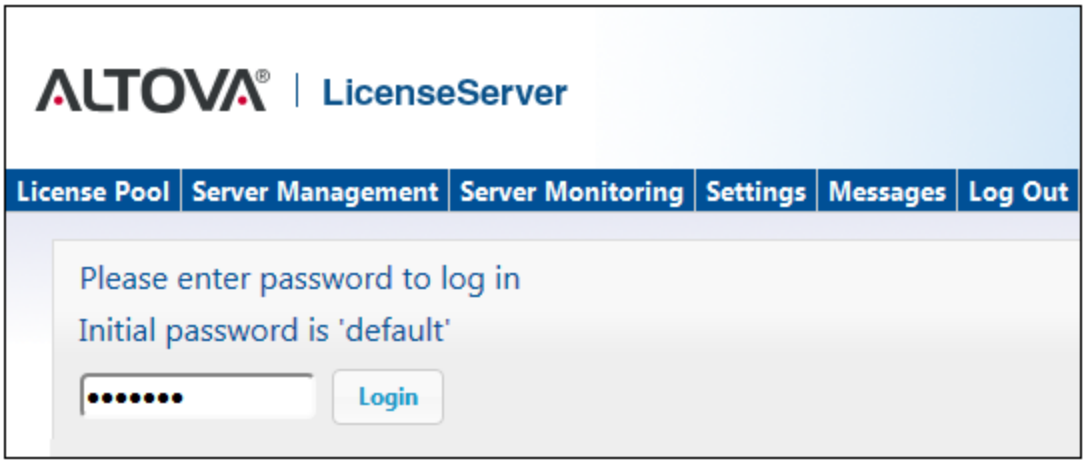
The URL is present in the HTML code of the Configuration page itself, which is named `WebUI.html` and is located at:

```
C:/ProgramData/Altova/LicenseServer/WebUI.html
```

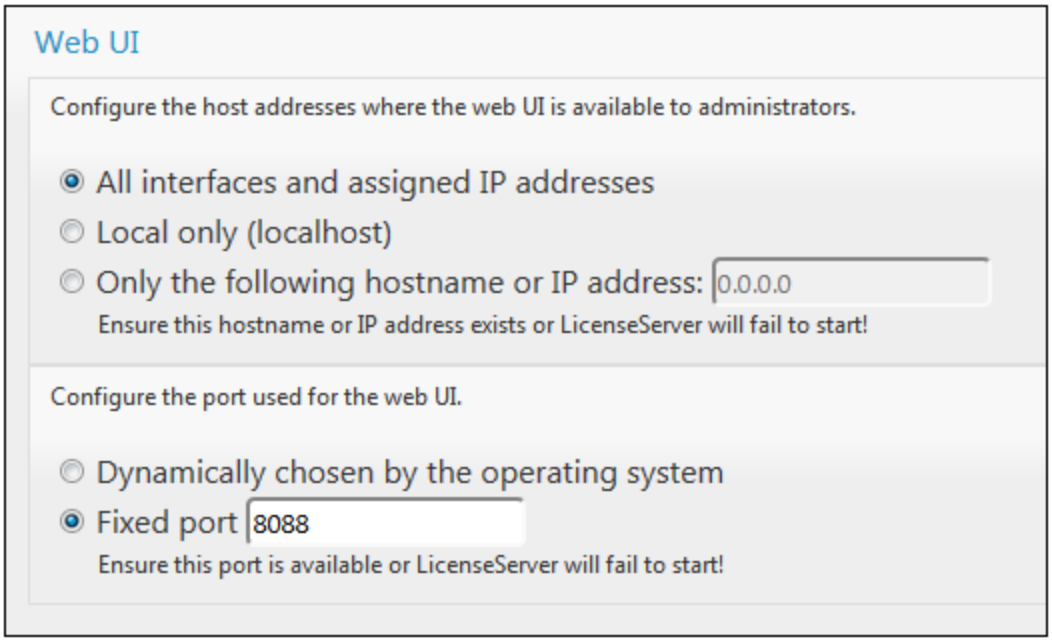
If you have set the URL of the Configuration page to be generated dynamically (in the Settings tab of the Configuration page), then a new URL is generated each time LicenseServer is started. You will need to check the current version of `WebUI.html` to find out the current URL of the Configuration page.

The dynamically generated URL in **`WebUI.html`** will have a form something like: `http://127.0.0.1:55541/optionally-an-additional-string`, and it is located in the function `checkIfServiceRunning()` in a script near the end of the `<head>` element. While the port number in the URL is dynamically assigned, the IP address part identifies the server on which LicenseServer has been installed. If you wish to access the LicenseServer Configuration page from another machine, make sure that the IP address part of the URL has the correct IP address or name of the server on which LicenseServer has been installed. For example, the URL could be something like: `http://MyServer:55541`.

Logging in with the initial password
After going through the steps above, the Configuration page is opened with the login mask displayed (*screenshot below*). You can log in with the initial password of `default`. After you have logged in, you can change your password in the Settings tab.



Setting a fixed or dynamic port for the Configuration page
The port of the Configuration page (and consequently its address) can be specified in the Settings page. By default the port is `8088`. You can set any other port you want for the LicenseServer Configuration page (*see screenshot below*). Alternatively, you allow the port to be selected dynamically each time LicenseServer starts up. In this case, you will need to find out the URL of the Configuration page from the file `WebUI.html` (see Open LicenseServer Config Page (Windows) and Open LicenseServer Config Page (Linux)).



The advantage of a fixed port is that the page URL is known in advance and therefore can be accessed easily. If the port is assigned dynamically, the port part of the URL will have to be looked up in the file `WebUI.html` each time LicenseServer is started afresh.

10.6.3 Open LicenseServer Config Page (Linux)

This section:

Opening the Configuration page for the first time with the returned URL
On Linux systems, when you register your Altova server product with LicenseServer via the CLI, the URL of the LicenseServer Configuration page is returned. On opening this URL in a browser, you are prompted to read and accept the license agreement. After accepting the license agreement, the Configuration page's login mask is displayed (*screenshot below*).

URL of the LicenseServer Configuration page
To open the LicenseServer Configuration page at any time, enter its URL in the address bar of a browser and press **Enter**. By default, the URL of the Configuration page will be:

```
http://<serverIPAddressOrName>:8088/
```

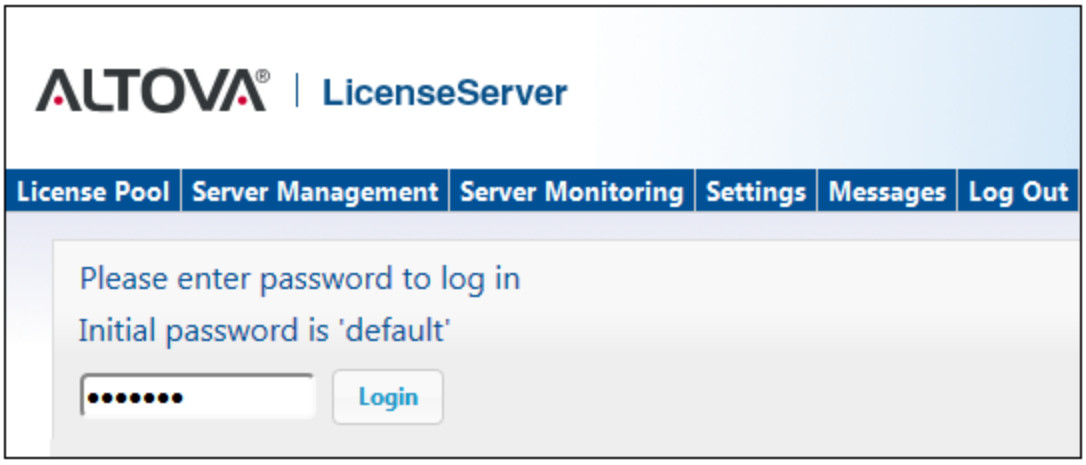
The URL is present in the HTML code of the Configuration page itself, which is named **webUI.html** and is located at:

```
/var/opt/Altova/LicenseServer/webUI.html
```

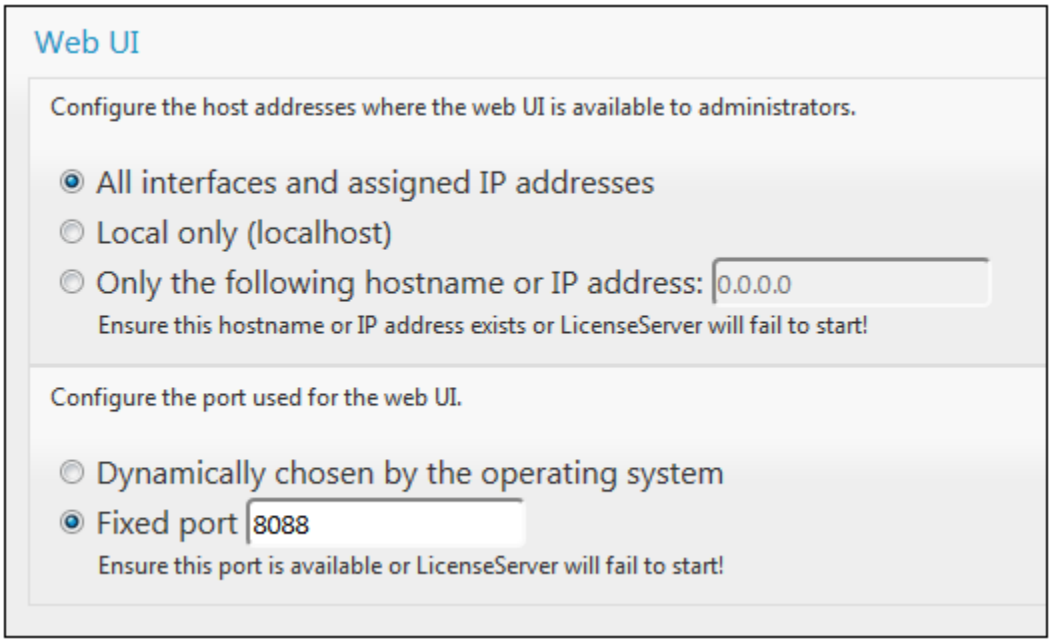
If you have set the URL of the Configuration page to be generated dynamically (in the Settings tab of the Configuration page), then a new URL is generated each time LicenseServer is started. You will need to check the current version of **webUI.html** to find out the current URL of the Configuration page.

The dynamically generated URL in **webUI.html** will have a form something like: `http://127.0.0.1:55541`, and it is located in the function `checkIfServiceRunning()` in a script near the end of the `<head>` element. While the port number in the URL is dynamically assigned, the IP address part identifies the server on which LicenseServer has been installed. If you wish to access the LicenseServer Configuration page from another machine, make sure that the IP address part of the URL has the correct IP address or name of the server on which LicenseServer has been installed. For example, the URL could be something like: `http://MyServer:55541`.

Logging in with the initial password
After going through the steps above, the Configuration page is opened with the login mask displayed (*screenshot below*). You can log in with the initial password of `default`. After you have logged in, you can change your password in the Settings tab.



Setting a fixed or dynamic port for the Configuration page
The port of the Configuration page (and consequently its address) can be specified in the Settings page. By default the port is `8088`. You can set any other port you want for the LicenseServer Configuration page (*see screenshot below*). Alternatively, you allow the port to be selected dynamically each time LicenseServer starts up. In this case, you will need to find out the URL of the Configuration page from the file `WebUI.html` (see Open LicenseServer Config Page (Windows) and Open LicenseServer Config Page (Linux)).



The advantage of a fixed port is that the page URL is known in advance and therefore can be accessed easily. If the port is assigned dynamically, the port part of the URL will have to be looked up in the file `WebUI.html` each time LicenseServer is started afresh.

10.6.4 Open LicenseServer Config Page (Mac OS X)

This section:

Opening the Configuration page for the first time with the returned URL
On Mac OS X systems, when you register your Altova server product with LicenseServer via the CLI, the URL of the LicenseServer Configuration page is returned. On opening this URL in a browser, you are prompted to read and accept the license agreement. After accepting the license agreement, the Configuration page's login mask is displayed (*screenshot below*).

URL of the LicenseServer Configuration page
To open the LicenseServer Configuration page at any time, enter its URL in the address bar of a browser and press **Enter**. By default, the URL of the Configuration page will be:

```
http://<serverIPAddressOrName>:8088/
```

The URL is present in the HTML code of the Configuration page itself, which is named **webUI.html** and is located at:

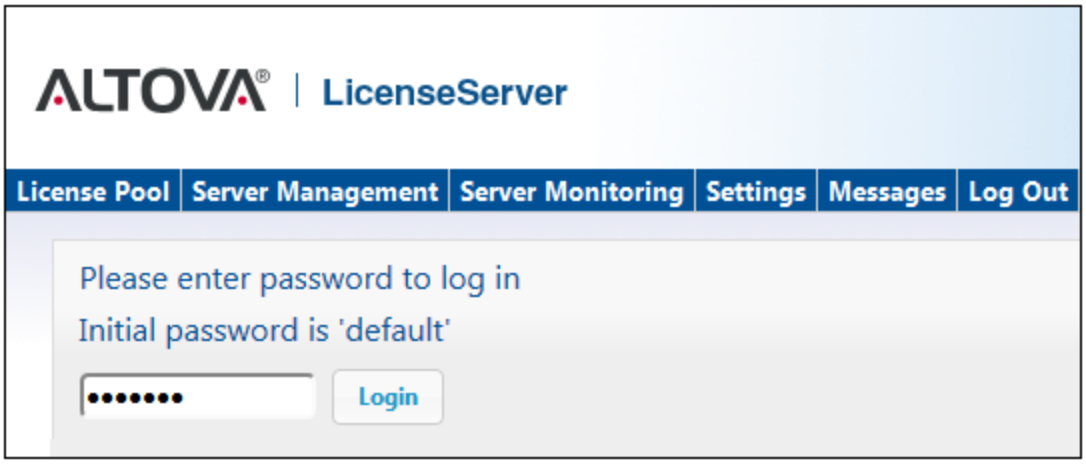
```
/var/Altova/LicenseServer/webUI.html
```

If you have set the URL of the Configuration page to be generated dynamically (in the Settings tab of the Configuration page), then a new URL is generated each time LicenseServer is started. You will need to check the current version of **webUI.html** to find out the current URL of the Configuration page.

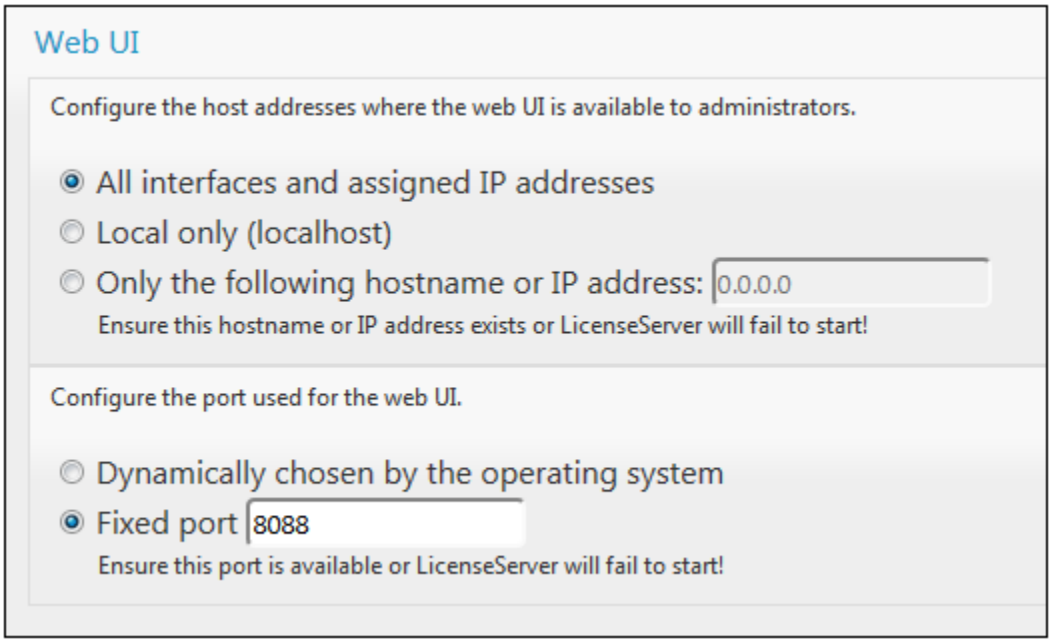
The dynamically generated URL in **webUI.html** will have a form something like: `http://127.0.0.1:55541`, and it is located in the function `checkIfServiceRunning()` in a script near the end of the `<head>` element. While the port number in the URL is dynamically assigned, the IP address part identifies the server on which LicenseServer has been installed. If you wish to access the LicenseServer Configuration page from another machine, make sure that the IP address part of the URL has the correct IP address or name of the server on which LicenseServer has been installed. For example, the URL could be something like: `http://MyServer:55541`.

Note: The Configuration page can also be accessed directly via the **Finder | Applications | Altova License Server** icon.

Logging in with the initial password
After going through the steps above, the Configuration page is opened with the login mask displayed (*screenshot below*). You can log in with the initial password of `default`. After you have logged in, you can change your password in the Settings tab.



Setting a fixed or dynamic port for the Configuration page
The port of the Configuration page (and consequently its address) can be specified in the Settings page. By default the port is `8088`. You can set any other port you want for the LicenseServer Configuration page (*see screenshot below*). Alternatively, you allow the port to be selected dynamically each time LicenseServer starts up. In this case, you will need to find out the URL of the Configuration page from the file `WebUI.html` (see Open LicenseServer Config Page (Windows) and Open LicenseServer Config Page (Linux)).



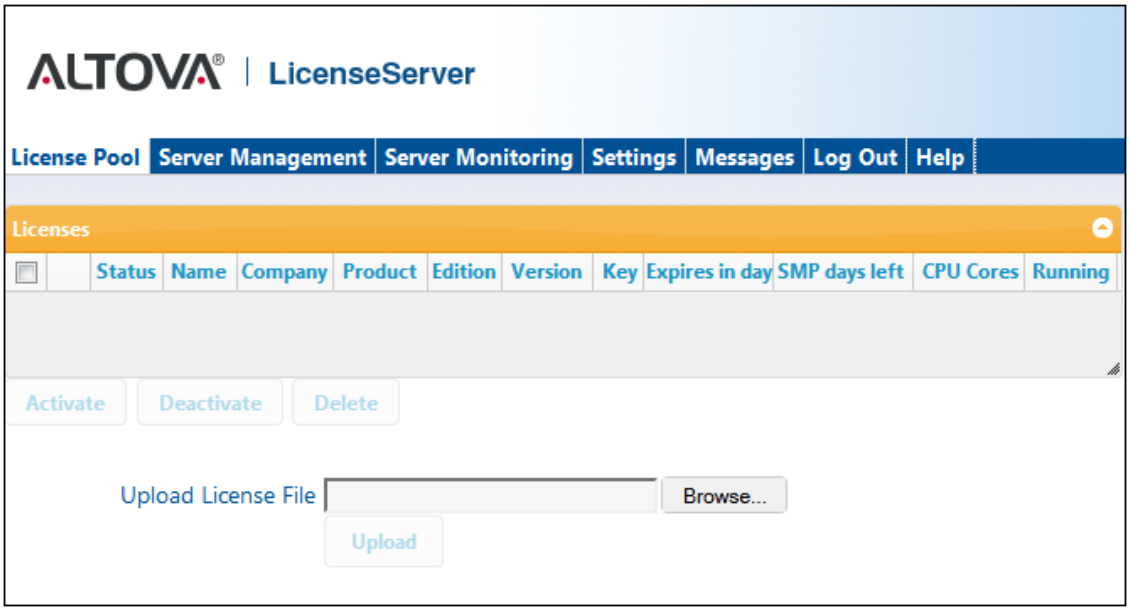
The advantage of a fixed port is that the page URL is known in advance and therefore can be accessed easily. If the port is assigned dynamically, the port part of the URL will have to be looked up in the file `WebUI.html` each time LicenseServer is started afresh.

10.6.5 Upload Licenses to LicenseServer

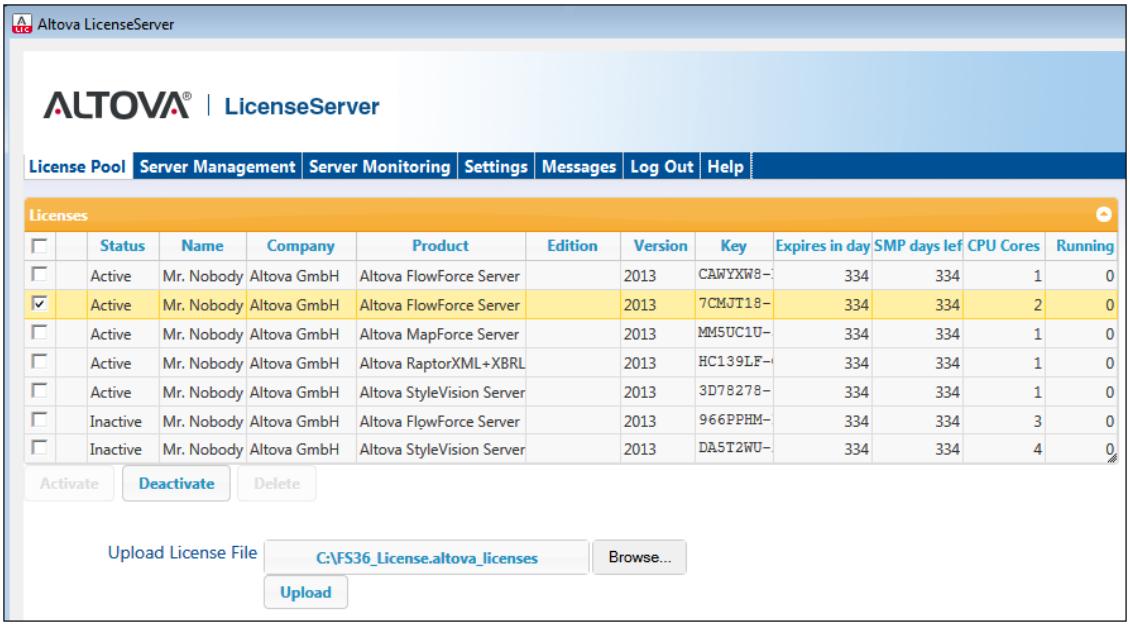
This section:

Uploading a license file to the license pool of LicenseServer
After you have obtained a license file from Altova, you must upload it to the Altova LicenseServer. (How to do this is described below.) Each license file can contain one or more licenses and depends on your purchase. When you upload a license file, all the licenses in it will be uploaded to the server and can be assigned to an Altova product that has been registered with that LicenseServer. All the uploaded licenses, from one or more license files and for all Altova products, are collected in a license pool on the LicenseServer. The license pool is displayed in the License Pool tab of the LicenseServer Configuration page (*screenshot below*).

License files are uploaded to the LicenseServer using the Upload function of the License Pool tab (*see screenshot below*).



Click the **Browse** button and select the license file you want. The license file will appear in the Upload License File text field and the **Upload** button will be enabled. Click the **Upload** button to upload the license file. All the licenses in the file are uploaded and displayed in the License Pool tab. The screenshot below shows multiple licenses, uploaded from multiple license files.



License status
License status values are as follows:

- *Activating:* When a license is uploaded into the license pool of LicenseServer, the server will transmit license-related data to the `altova.com` master licensing server to validate, authenticate, and activate the license that was supplied. This is necessary to ensure compliance with the Altova license agreements. During this initial activation and authentication transaction—which typically lasts between 30 seconds and a couple of minutes, depending on your Internet connection, speed, and overall network traffic—the status of the license will be indicated as *Activating...*.

- *Failed Verification:* If a connection with the `altova.com` master licensing server cannot be made, then the status of the license in the pool will be shown as *Failed Verification*. If this happens, check your Internet connection and firewall rules to ensure that LicenseServer is able to communicate with the `altova.com` master licensing server.

- *Active:* Once the license has been authenticated and activated, the status in the pool will change to *Active*.

- *Inactive:* If a license has been verified, but is present on another LicenseServer on the network, the status in the pool will be shown as *Inactive*. An *Inactive* status also results when a license is manually deactivated in the license pool by the administrator.

- *Blocked:* A license is shown in the license pool as *Blocked* if there was a problem authenticating the license and the `altova.com` master licensing server has not granted permission to the LicenseServer to use this license. This could be the result of a license agreement violation, over-usage of a license, or other compliance issues. Should you see a license showing up as *Blocked*, please contact Altova Support with your license information and any other relevant data.

These statuses are summarized in the table below:

Status	Meaning
Activating...	On upload, license information is sent to `altova.com` for verification. Refresh the browser to view the updated status. Verification and activation can take a few minutes.
Failed Verification	A connection to `altova.com` could not be made. After establishing a connection, either restart the service or activate the license (with the **Activate** button).
Active	Verification was successful, the license is active.
Inactive	Verification was successful, but the license is on another LicenseServer on the network. Licenses can be made inactive with the **Deactivate** button.
Blocked	Verification was not successful. License is invalid and is blocked. Contact Altova Support.

Note: After a license has been sent to `altova.com` for verification, the browser must be refreshed to see the updated status. Verification and activation can take a few minutes.

Note: If a connection to `altova.com` could not be made, the status will be *Failed Verification*. After establishing a connection, either restart the service or try activating the license with the **Activate** button.

Note: When a license is given a status of *Inactive* or *Blocked*, a message explaining the status is also added to the Messages log.

Only an active license can be assigned to a product installation. An inactive license can be activated or deleted from the license pool. If a license is deleted from the license pool, it can be uploaded again to the pool by uploading the license file containing it. When a license file is updated, only those licenses in it that are not already in the pool will be uploaded to the pool. To activate, deactivate, or delete a license, select it and then click the **Activate**, **Deactivate**, or **Delete** button, respectively.

Activate the license/s you wish to use
Before you can assign a license to an Altova product, it must be active. So do ensure it is active. If it is inactive, select it and click **Activate**.

Next Steps
After you have uploaded the license file to the LicenseServer and checked that the license you want is active, do the following:

1. Register the Altova server product (FlowForce Server, MapForce Server, StyleVision Server) with LicenseServer. (If you have already done this prior to uploading the license file, you can now start assigning licenses.)
2. Assign a license to your Altova product that has been registered with the LicenseServer.

10.6.6 Register FlowForce Server with LicenseServer

This section:

Methods of registering FlowForce Server
FlowForce Server can be registered with LicenseServer using any of the following methods:

- Via the FlowForce Server Setup page (Windows and Linux)
- Via the FlowForce CLI (Windows)
- Via the FlowForce CLI (Linux)

Accessing the FlowForce Server Setup page (Windows)
The FlowForce Server Setup page can be accessed in one of the following ways:

- Via the **Start** menu:
  Start | *Altova FlowForce Server 2013* | *FlowForce Server Setup Page*
- Via Altova ServiceController: Click the ServiceController icon in the system tray. In the menu that pops up, select *Altova FlowForce Web* | *Setup*.

This pops up the FlowForce Server Setup page (*screenshot above*).

Accessing the FlowForce Server Setup page (Linux)
After you have installed FlowForce Server on Linux (see the FlowForce Server user documentation for information about how to do this), start FlowForce Web Server as a service with the following command:

```
sudo /etc/init.d/flowforcewebserver start
```

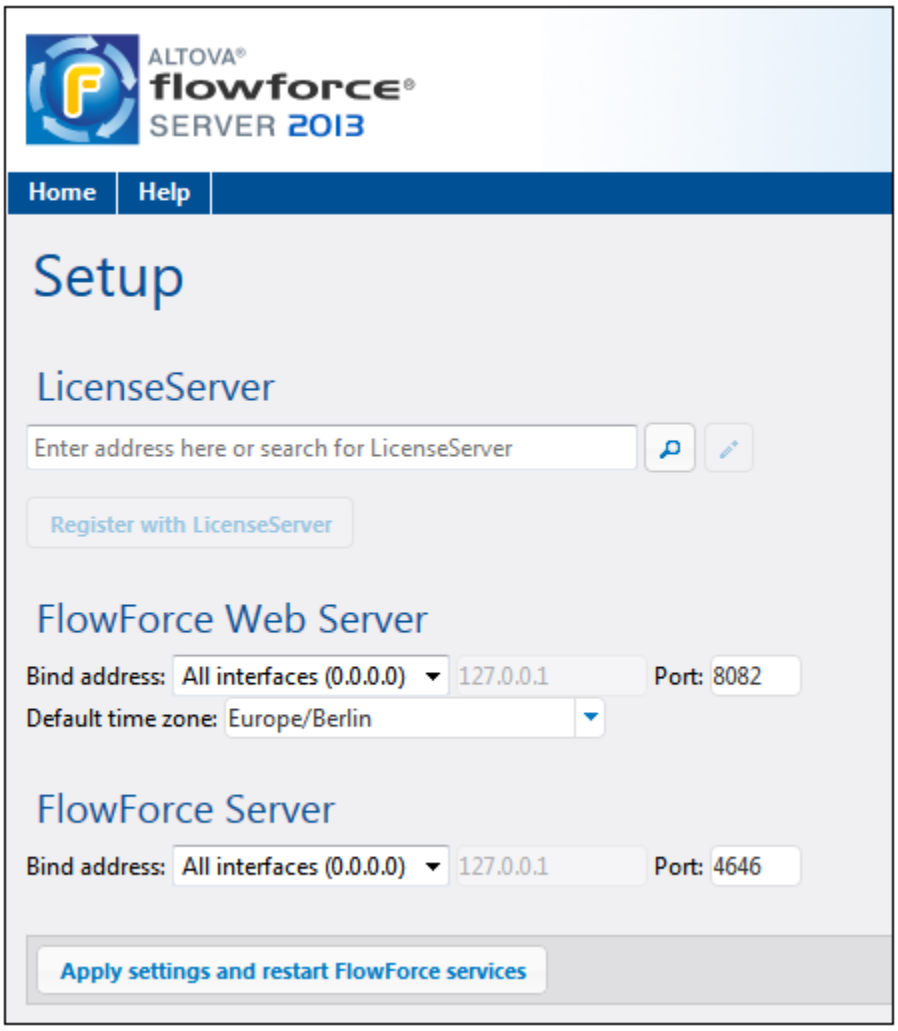
A message containing the URL of the FlowForce Server Setup appears in the terminal window:

FlowForceWeb running on `http://127.0.1.1:3459/setup?key=52239315203`

Enter the URL into the address field of a browser and hit Enter to access FlowForce Server Setup page (*screenshot above*).

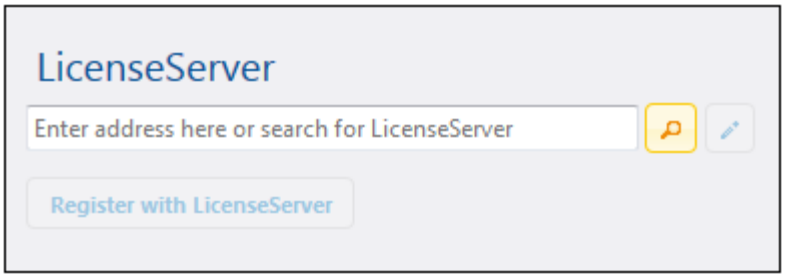
Registering FlowForce Server via the Setup page (Windows and Linux)
In the Setup page (*screenshot below*)—how to access it is described above—the LicenseServer

field specifies the Altova LicenseServer to be used for registration.

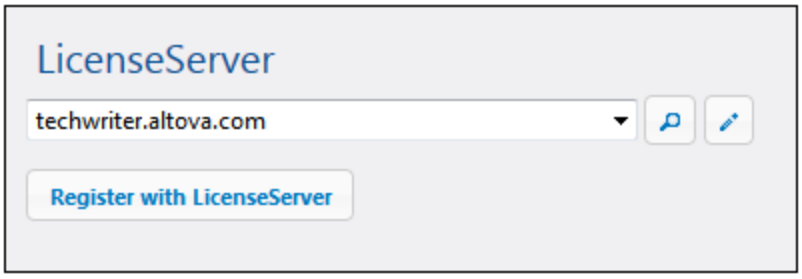


The LicenseServer can be specified in one of two ways.

- You can search for Altova LicenseServers that are currently available on the network—that is, those that are currently running. Do this by clicking the **Search for Altova LicenseServers** button (*highlighted yellow in the screenshot below*).

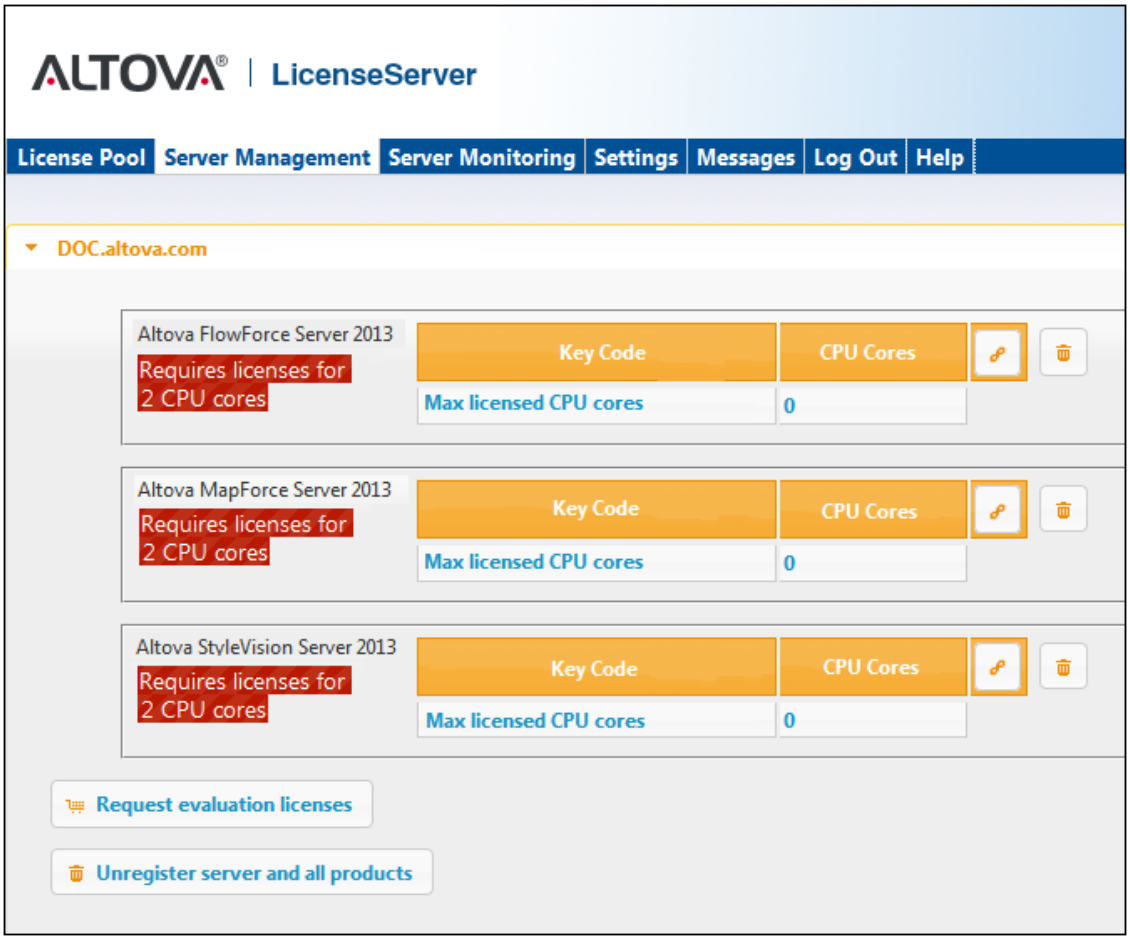
  The search returns a list of available Altova LicenseServers on the network. One LicenseServer will be selected (*screenshot below*) and the others will be available in the dropdown list of the combo box. Select the LicenseServer on which your FlowForce license is stored.



- Alternatively, you can enter the address of the LicenseServer in the LicenseServer field. If the currently running LicenseServers are available as a dropdown list, you must click the **Manually Enter Address** button to be able to enter an address in the LicenseServer field.

After you have specified the LicenseServer, click **Register with LicenseServer**. The Altova server application will be registered with the specified LicenseServer, and that LicenseServer's Configuration page will open in a browser with its Server Management tab active (*screenshot below*).

Note: You may need to allow pop-ups in order for the LicenseServer Configuration page to be displayed.



In the screenshot below, three Altova products have been registered with the Altova LicenseServer at DOC.altova.com. How to assign licenses is described in the next section, Assign Licenses to Registered Products.

Registering FlowForce Server via the FlowForce CLI (Windows)

On Windows machines, FlowForce Server can also be registered with an Altova LicenseServer on your network via the command line (CLI) by using the licenseserver command:

```
FlowForceServer licenseserver Server-Or-IP-Address
```

For example, if LicenseServer is running on http://localhost:8088, then register FlowForce Server with:

```
FlowForceServer licenseserver localhost
```

If FlowForce Server was installed with other Altova server products as sub-packages, registering FlowForce Server will automatically also register the Altova server products. After successfully registering FlowForce Server, you can go to LicenseServer and assign a license to FlowForce Server. How to do this is described in the section Assign Licenses to Registered Products.

Registering FlowForce Server via the FlowForce CLI (Linux)

On Linux machines, FlowForce Server can be registered with LicenseServer by using the licenseserver command of the FlowForce Server CLI. Note that FlowForce Server must be started with root rights.

```
sudo /opt/Altova/FlowForceServer2013/bin/flowforceserver licenseserver
localhost
```

In the command above, localhost is the name of the server on which LicenseServer is installed. Notice also that the location of the FlowForce Server executable is:

```
/opt/Altova/MapForceServer2013/bin
```

After successfully registering FlowForce Server, you can go to LicenseServer and assign a license to FlowForce Server. How to do this is described in the section Assign Licenses to Registered Products.

Next Steps

After you have registered your Altova product with LicenseServer, do the following:

1. If you have not already uploaded your license file/s to the LicenseServer (see previous section, Upload Licenses to LicenseServer), upload the license file now and check that the license you want is active. If you have already done this, carry on to the next step, Assign Licenses.
2. Assign a license to your Altova product that has been registered with the LicenseServer.

10.6.7 Register MapForce Server with LicenseServer

This section:

MapForce Server can be installed as part of the FlowForce Server package or as a standalone server product. In either case, it must be registered with Altova LicenseServer. Only after it has been registered with LicenseServer can a license be assigned to it from LicenseServer. On Windows systems, if MapForce Server was installed as part of the FlowForce Server package, it will automatically be registered when FlowForce is registered. On Linux systems, only if MapForce Server is installed after FlowForce Server will it be registered automatically when FlowForce Server is registered subsequently.

Registering MapForce Server from FlowForce Server (Windows)
MapForce Server is packaged with FlowForce Server, so when FlowForce Server is registered with an Altova LicenseServer on your network, MapForce Server will automatically also be registered with LicenseServer. How to register FlowForce Server is described in the FlowForce Server documentation and in the section, Register FlowForce Server with LicenseServer.

After the registration, you can go to LicenseServer and assign a MapForce Server license to MapForce Server. How to do this is described in the section, Assign Licenses to Registered Products.

Registering a standalone MapForce Server (Windows)
If you have installed MapForce Server as a standalone package, you must register it with an Altova LicenseServer on your network and then license it from the Altova LicenseServer. You can register MapForce Server via its command line interface (CLI) by using the `licenseserver` command:

```
MapForceServer licenseserver Server-Or-IP-Address
```

For example, if LicenseServer is running on `http://localhost:8088`, then register MapForce Server with:

```
MapForceServer licenseserver localhost
```

After successfully registering MapForce Server, you can go to LicenseServer and assign a license to MapForce Server. How to do this is described in the section, Assign Licenses to Registered Products.

Registering MapForce Server (Linux)
On Linux machines, MapForce Server can be registered with LicenseServer by using the

`licenseserver` command of the MapForce Server CLI. Note that MapForce Server must be started with root rights.

```
sudo /opt/Altova/MapForceServer2013/bin/mapforceserver licenseserver
localhost
```

In the command above, `localhost` is the name of the server on which LicenseServer is installed. Notice also that the location of the MapForce Server executable is:

```
/opt/Altova/MapForceServer2013/bin
```

After successfully registering MapForce Server, you can go to LicenseServer and assign a license to MapForce Server. How to do this is described in the section Assign Licenses to Registered Products.

Next Steps

After you have registered your Altova product with LicenseServer, do the following:

1. If you have not already uploaded your license file/s to the LicenseServer (see previous section, Upload Licenses to LicenseServer), upload the license file now and check that the license you want is active. If you have already done this, carry on to the next step, Assign Licenses.
2. Assign a license to your Altova product that has been registered with the LicenseServer.

10.6.8 Register RaptorXML(+XBRL) Server with LicenseServer

This section:

RaptorXML(+XBRL) Server must be installed on the server machine and then be started as a service. It must then be registered with LicenseServer. Only after it has been registered with LicenseServer can a license be assigned to it from LicenseServer. This section describes how to register RaptorXML(+XBRL) Server with LicenseServer.

Registering RaptorXML(+XBRL) Server (Windows)

You can register RaptorXML(+XBRL) Server via its command line interface (CLI) by using the `licenseserver` command:

```
RaptorXML licenseserver Server-Or-IP-Address
```

For example, if LicenseServer is running on `http://localhost:8088`, then register RaptorXML(+XBRL) Server with:

```
RaptorXML licenseserver localhost
```

After successfully registering RaptorXML(+XBRL) Server, you can go to LicenseServer and assign a license to RaptorXML(+XBRL) Server. How to do this is described in the section Assign Licenses to Registered Products.

Registering RaptorXML(+XBRL) Server (Linux)

On Linux machines, RaptorXML(+XBRL) Server can be registered with LicenseServer by using the `licenseserver` command of the RaptorXML(+XBRL) Server CLI. Note that RaptorXML (+XBRL) Server must be started with root rights.

```
sudo /opt/Altova/RaptorXMLServer2013/bin/raptorxmlserver licenseserver
localhost
```

In the command above, `localhost` is the name of the server on which LicenseServer is installed. Notice also that the location of the RaptorXML(+XBRL) Server executable is:

```
/opt/Altova/RaptorXMLServer2013/bin
```

After successfully registering RaptorXML(+XBRL) Server, you can go to LicenseServer and assign a license to RaptorXML(+XBRL) Server. How to do this is described in the section Assign Licenses to Registered Products.

Next Steps

After you have registered your Altova product with LicenseServer, do the following:

1. If you have not already uploaded your license file/s to the LicenseServer (see previous section, Upload Licenses to LicenseServer), upload the license file now and check that the license you want is active. If you have already done this, carry on to the next step, Assign Licenses.
2. Assign a license to your Altova product that has been registered with the LicenseServer.

10.6.9 Register StyleVision Server with LicenseServer

This section:

StyleVision Server can be installed as part of the FlowForce Server package or as a standalone server product. In either case, it must be registered with Altova LicenseServer. Only after it has been registered with LicenseServer can a license be assigned to it from LicenseServer. On Windows systems, if StyleVision Server was installed as part of the FlowForce Server package, it will automatically be registered when FlowForce is registered. On Linux systems, only if StyleVision Server is installed after FlowForce Server will it be registered automatically when FlowForce Server is registered subsequently.

Registering StyleVision Server from FlowForce (Windows)
StyleVision Server is packaged with FlowForce Server, so when FlowForce Server is registered with an Altova LicenseServer on your network, StyleVision Server will automatically also be registered with LicenseServer. How to register FlowForce Server is described in the FlowForce Server documentation and in the section, Register FlowForce Server with LicenseServer.

After the registration, you can go to LicenseServer and assign a StyleVision Server license to StyleVision Server. How to do this is described in the section Assign Licenses to Registered Products.

Registering a standalone StyleVision Server (Windows)
If you have installed StyleVision Server as a standalone package on Windows, you must register it with an Altova LicenseServer on your network and then license it from the Altova LicenseServer. You can register StyleVision Server via its command line interface (CLI) by using the `licenseserver` command:

```
StyleVisionServer licenseserver Server-Or-IP-Address
```

For example, if LicenseServer is running on `http://localhost:8088`, then register StyleVision Server with:

```
StyleVisionServer licenseserver localhost
```

After successfully registering StyleVision Server, you can go to LicenseServer and assign a license to StyleVision Server. How to do this is described in the section Assign Licenses to Registered Products.

Registering StyleVision Server (Linux)
On Linux machines, StyleVision Server can be registered with LicenseServer by using the

`licenseserver` command of the StyleVision Server CLI. Note that StyleVision Server must be started with root rights.

```
sudo /opt/Altova/StyleVisionServer2013/bin/stylevisionserver licenseserver
localhost
```

In the command above, `localhost` is the name of the server on which LicenseServer is installed. Notice also that the location of the StyleVision Server executable is:

```
/opt/Altova/StyleVisionServer2013/bin
```

After successfully registering StyleVision Server, you can go to LicenseServer and assign a license to StyleVision Server. How to do this is described in the section Assign Licenses to Registered Products.

Next Steps

After you have registered your Altova product with LicenseServer, do the following:

1. If you have not already uploaded your license file/s to the LicenseServer (see previous section, Upload Licenses to LicenseServer), upload the license file now and check that the license you want is active. If you have already done this, carry on to the next step, Assign Licenses.
2. Assign a license to your Altova product that has been registered with the LicenseServer.

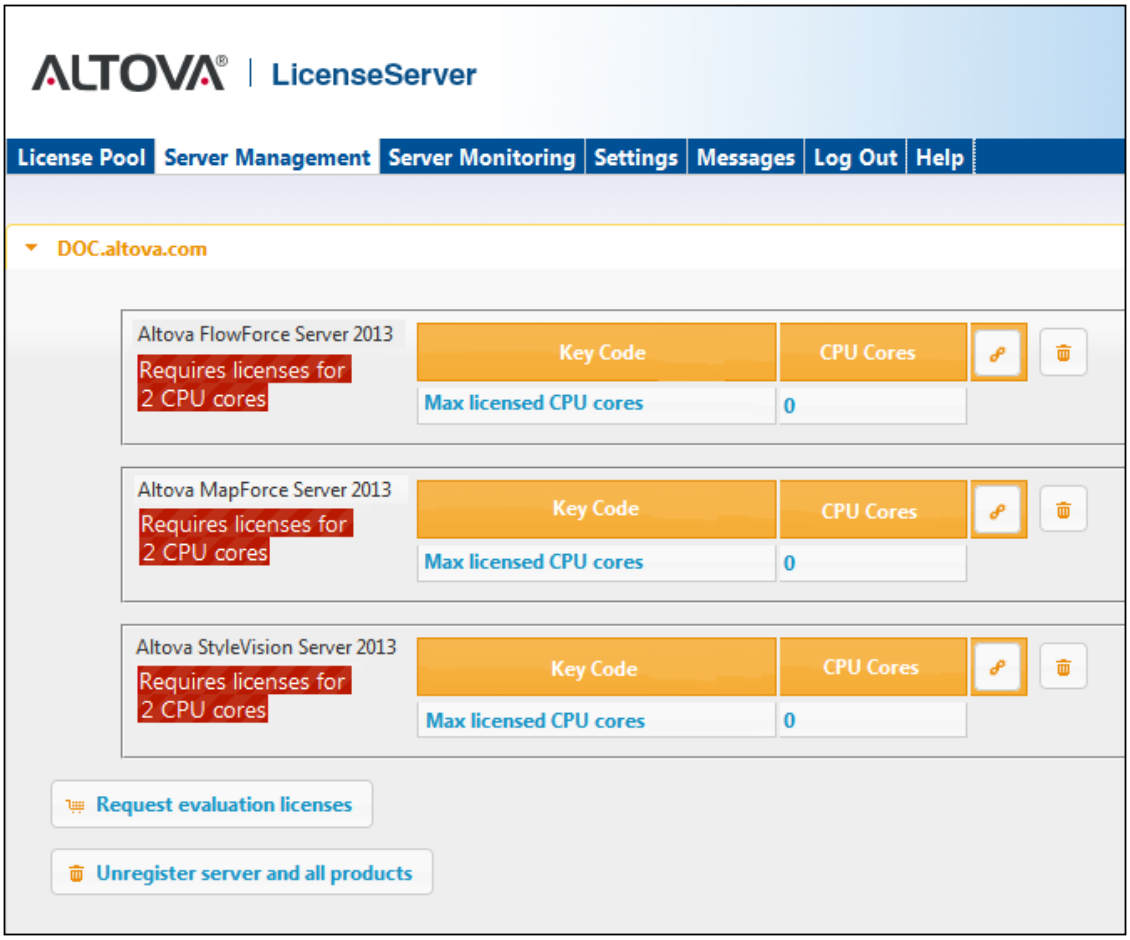
10.6.10 Assign Licenses to Registered Products

This section:

Before assigning a license
Before you assign a license to an Altova product, make sure that:

- The relevant license has been uploaded to the license pool of LicenseServer and that the license is active.
- Your Altova product has been registered with LicenseServer.

The Server Management tab
Licenses are assigned in the Server Management tab of the LicenseServer Configuration page (*screenshot below*). The screenshot shows that three Altova products have been registered with LicenseServer. (Since MapForce Server and StyleVision Server are bundled with FlowForce Server, registering FlowForce Server with LicenseServer automatically also registers MapForce Server and StyleVision Server. No additional registration of the latter two products are required if FlowForce Server is registered.)



Note the following points about the Server Management tab:

- Each product is listed under the name of its client machine. In the screenshot above, one client machine, named `Doc.altova.com`, is listed. This client machine (`Doc.altova.com`) has three Altova products registered with the LicenseServer. If an Altova product on a different client machine is registered with this LicenseServer, then that client machine, with its registered products, will also be listed in the Server Management tab.
- Each registered Altova product on a client machine has its own *Key Code* entry, which takes the key code of a license. A registered product's key code is assigned by clicking its **Edit Assigned Licenses** button (*see icon list below*) and selecting the required license from those available for that product (for example, FlowForce Server) in the license pool. This procedure is explained in more detail below.
- Each product also has a line stating how many CPU cores need to be licensed to run that product on that client. If the number of licensed cores is less than the number required, then the information is marked in red (*see screenshot above*). (The number of CPU cores that need to be licensed is the number of CPU cores on that client and is obtained from the client machine by LicenseServer.)

Icons in the Server Management tab

Edit Assigned Licenses. Available with each product. Pops up the Manage Licenses dialog, in which new licenses can be assigned to the product and already assigned licenses can be edited.

Show Licenses. Appears with each license. Switches to the License Pool tab and highlights the selected license, so that license details can be read.

Unregister This Product. Available with each product. The selected product (on the selected client machine) will be unregistered from LicenseServer.

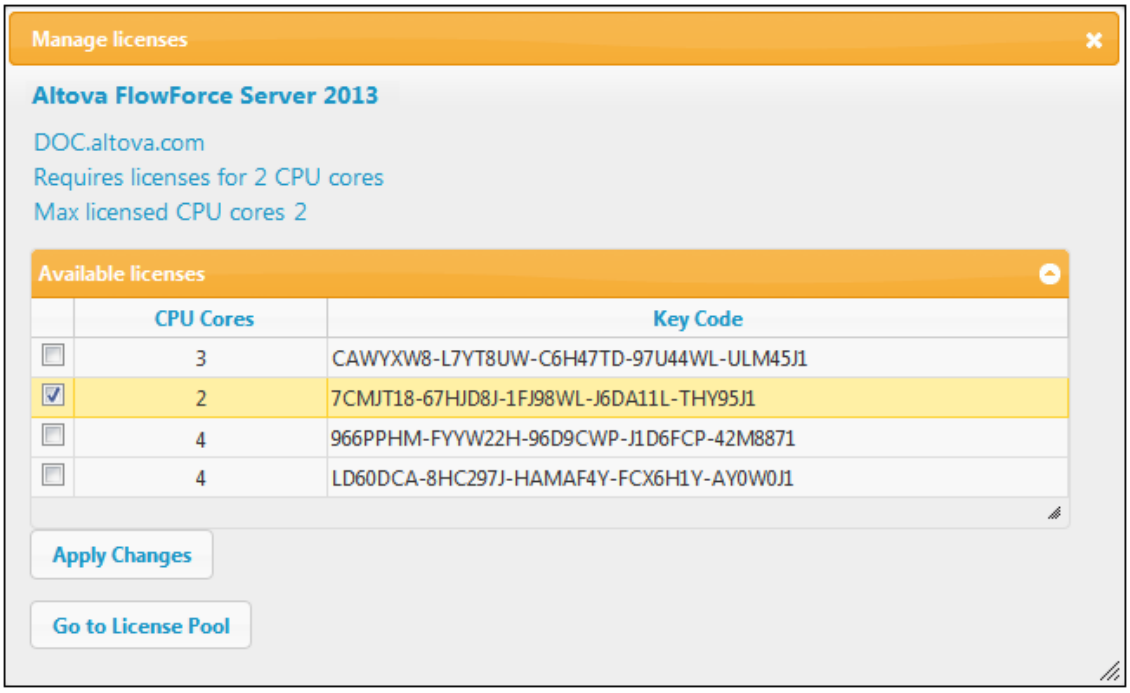
Note on cores and licenses
The licensing of Altova server products is based on the number of processor cores available on the product machine. For example, a dual-core processor has two cores, a quad-core processor four cores, a hexa-core processor six cores, and so on. The number of cores licensed for a product on a particular server machine must be greater than or equal to the number of cores available on that server, whether it's a physical or virtual machine.

For example, if a server has eight cores (an octa-core processor), you must purchase at least an 8-core license. You can also combine licenses to achieve the core count. So, two 4-core licenses can also be used for an octa-core server instead of an 8-core license.

If you are using a computer server with a large number of CPU cores but only have a low volume to process, you may also create a virtual machine that is allocated a smaller number of cores, and purchase a license for that number. Such a deployment, of course, will have less processing speed than if all available cores on the server were utilized.

Note: Each license can be used for only one client machine at a time, even if it has unused licensing capacity. For example, if a 10-core license is used for a client machine that has 6 CPU cores, then the remaining 4 cores of the license cannot be used simultaneously for another client machine.

Assigning a license
To assign a license to a registered product, click the **Edit Assigned Licenses** button of that product. This pops up the Manage Licenses dialog (*screenshot below*).

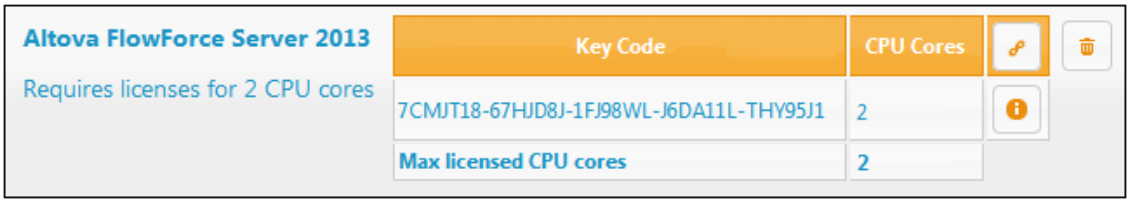


Note the following points about the licenses displayed in the Manage Licenses dialog:

- The product to be licensed is listed at the top left of the dialog. In the screenshot above the product is Altova FlowForce Server 2013.
- The dialog displays all the currently active licenses for that product in the license pool. In our screenshot, four currently active FlowForce Server licenses are in the license pool. LicenseServer will automatically detect from each license in the pool the product for which it has been issued.
- The licenses in the screenshot above have been licensed, respectively, for 3 CPU cores, 2 CPU cores, 4 CPU cores, and 4 CPU cores.
- You need to know the number of processor cores on the server on which the Altova server product has been installed. If the machine has a dual-core processor, you need a two-core (the CPU Cores count) license. This license could be, for example, the second license in the list shown in the screenshot above. You can also combine licenses. So, if the machine's processor is octa-core (eight-core), you can combine two 4-core licenses; for example, the third and fourth licenses in the list shown in the screenshot above.
- The Manage Licenses dialog will list only currently active licenses for that product. Licenses for other Altova products will not be listed.
- Licenses that have been assigned already—for example, to another installation of the product on the network—will have their check boxes checked. So only unchecked licenses may be selected.
- CPU cores indicates for how many CPU cores a license is valid.
- If you wish to make modifications to the license pool—for example, to upload, activate, deactivate, or delete a license—click the **Go to License Pool** button.

Select the license you wish to assign. The license's check box will be checked. Also, the total number of CPU cores licensed for that product on that client is listed near the top left of the dialog as *Max licensed CPU cores (see screenshot above)*. You can select more licenses if you wish to increase the number of licensed CPU cores for that product on that client. The *Max licensed CPU cores* in this case will be the sum of the CPU cores of all the selected licenses.

After selecting the license/s, click **Apply Changes**. The license/s will be assigned to that product and displayed in the Server Management tab (*see screenshot below*). The screenshot below shows that a 2-CPU-core license for Altova FlowForce Server has been assigned (to the client machine `Doc.altova.com`).



Unregistering products
Each Altova product registered with LicenseServer is listed in the Server Management tab under its client machine name and has an **Unregister** icon to its right. Click this icon to unregister the product. If a license was assigned to the product, the assignment will be terminated when the product is unregistered. To unregister all products, click the **Unregister Server and All Products** button at the bottom of the Server Management tab (*see first screenshot in this section*).

To re-register a product, go to the product's pre-configuration page.

10.7 Configuration Page Reference

The LicenseServer Configuration page is the administrator's interface with LicenseServer. It allows the management of LicenseServer and the licensing of Altova products that have been r egistered with LicenseServer (FlowForce Server, MapForce Server, StyleVision Server).

The LicenseServer Configuration page is opened via a web browser. How to open the Configuration page is described in the sections, Open LicenseServer Config Page (Windows) and Open LicenseServer Config Page (Linux).

This section is a user's reference for the Configuration page and is organized by the tabs of the Configuration page:

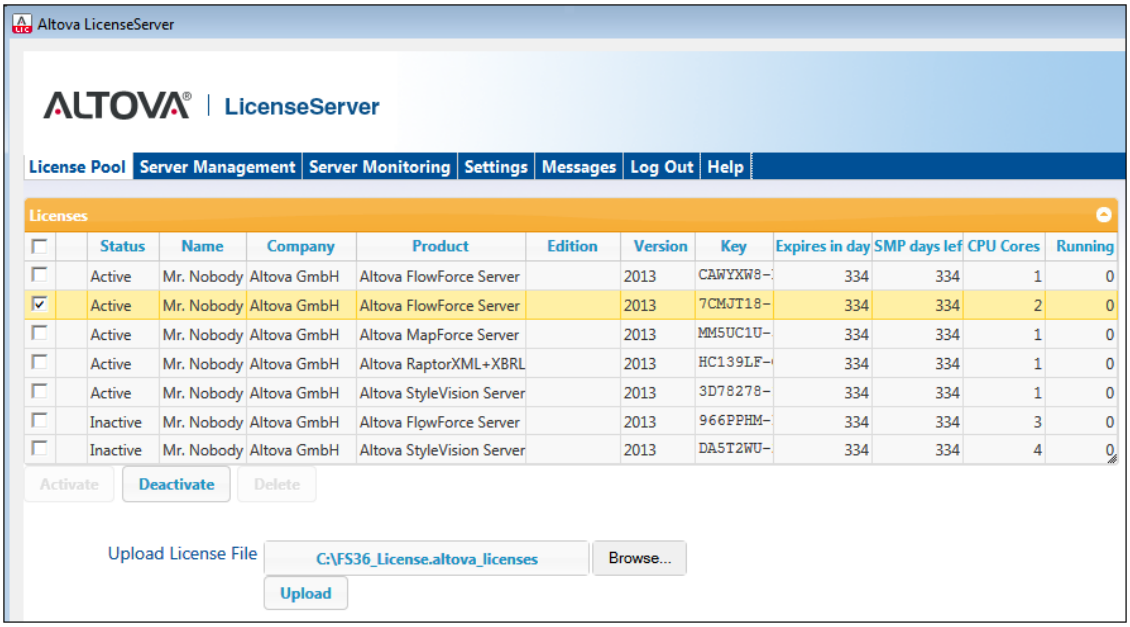
- License Pool
- Server Management
- Server Monitoring
- Server Monitoring
- Messages, Log Out

For a step-by-step guide of how to assign licenses with LicenseServer, see the section How to Assign Licenses.

10.7.1 License Pool

The **License Pool** tab displays all the licenses that are currently on the LicenseServer (*see screenshot below*). When a license file is uploaded to the LicenseServer with the **Upload** button on this page, all the licenses contained in the license file are placed in the license pool on the server and are displayed on the License Pool page.

The License Pool page displays information about all the licenses currently on the LicenseServer and thus provides a convenient overview of all Altova product licenses. On this page you can also activate, deactivate, and delete selected licenses.



Uploading a license
To upload a license file (which you receive from Altova GmbH for your Altova product), click the **Browse** button, browse for the license file and select it. On clicking **Upload**, all the licenses contained in the license file are placed in the license pool and displayed on the License Pool page (*screenshot above*).

License status
License status values are as follows:

- *Activating:* When a license is uploaded into the license pool of LicenseServer, the server will transmit license-related data to the `altova.com` master licensing server to validate, authenticate, and activate the license that was supplied. This is necessary to ensure compliance with the Altova license agreements. During this initial activation and authentication transaction—which typically lasts between 30 seconds and a couple of minutes, depending on your Internet connection, speed, and overall network traffic—the status of the license will be indicated as *Activating...*.

- *Failed Verification:* If a connection with the `altova.com` master licensing server cannot

be made, then the status of the license in the pool will be shown as *Failed Verification*. If this happens, check your Internet connection and firewall rules to ensure that LicenseServer is able to communicate with the `altova.com` master licensing server.

- *Active:* Once the license has been authenticated and activated, the status in the pool will change to *Active*.
- *Inactive:* If a license has been verified, but is present on another LicenseServer on the network, the status in the pool will be shown as *Inactive*. An *Inactive* status also results when a license is manually deactivated in the license pool by the administrator.
- *Blocked:* A license is shown in the license pool as *Blocked* if there was a problem authenticating the license and the `altova.com` master licensing server has not granted permission to the LicenseServer to use this license. This could be the result of a license agreement violation, over-usage of a license, or other compliance issues. Should you see a license showing up as *Blocked*, please contact Altova Support with your license information and any other relevant data.

These statuses are summarized in the table below:

Status	Meaning
Activating...	On upload, license information is sent to `altova.com` for verification. Refresh the browser to view the updated status. Verification and activation can take a few minutes.
Failed Verification	A connection to `altova.com` could not be made. After establishing a connection, either restart the service or activate the license (with the **Activate** button).
Active	Verification was successful, the license is active.
Inactive	Verification was successful, but the license is on another LicenseServer on the network. Licenses can be made inactive with the **Deactivate** button.
Blocked	Verification was not successful. License is invalid and is blocked. Contact Altova Support.

Note: After a license has been sent to `altova.com` for verification, the browser must be refreshed to see the updated status. Verification and activation can take a few minutes.

Note: If a connection to `altova.com` could not be made, the status will be *Failed Verification*. After establishing a connection, either restart the service or try activating the license with the **Activate** button.

Note: When a license is given a status of *Inactive* or *Blocked*, a message explaining the status is also added to the Messages log.

Only an active license can be assigned to a product installation. An inactive license can be activated or deleted from the license pool. If a license is deleted from the license pool, it can be uploaded again to the pool by uploading the license file containing it. When a license file is updated, only those licenses in it that are not already in the pool will be uploaded to the pool. To activate, deactivate, or delete a license, select it and then click the **Activate**, **Deactivate**, or **Delete** button, respectively.

Connection to the Master Licensing Server at altova.com
The Altova LicenseServer needs to be able to communicate with the Master Licensing Server at `altova.com` to validate and authenticate license-related data and to ensure continuous compliance with the Altova license agreements. This communication occurs over HTTPS using port 443. If the Altova LicenseServer, after making the initial verification with the `altova.com` Master Licensing Server, is unable to again connect with `altova.com` for a duration of more than 5 days (= 120 hours), then the Altova LicenseServer will no longer permit the usage of any Altova software products connected to the Altova LicenseServer.

Any such loss of connection with the `altova.com` master servers will be logged in the Messages tab of the Configuration page of the Altova LicenseServer. In addition, the administrator can configure the Altova LicenseServer to automatically send an alert email when the connection to `altova.com` is lost. Alert Mail settings are available in the Settings tab of the Configuration page.

Activating, deactivating, and deleting a license
An active license can be deactivated by selecting the license and clicking **Deactivate**. An inactive license can be activated (**Activate** button) or deleted (**Delete** button). When a license is deleted it is removed from the license pool. A deleted license can be added again to the license pool by uploading the license file containing it. If a license file is re-uploaded, only licenses that are not already in the license pool will be added to the license pool; licenses that are already in the pool will not be re-added.

License information
The following license information is displayed:

- *Status:* There are five values: *Failed Verification* | *Activating* | *Active* | *Inactive* | *Blocked*. See License status above.
- *Name, Company:* The name and company of the licensee. This information was submitted at the time of purchase.
- *Product, Edition, Version:* The version and edition of the licensed products.
- *Key, Expires in days, SMP (days left):* The license key to unlock the product, and the number of days left before the license expires. Each licensed purchase comes with a Support & Maintenance Package, which is valid for a certain number of days. The *SMP* column notes how many SMP days are still left.
- *CPU Cores:* The number of CPU cores that the license allows.
- *Running:* The number of CPU cores currently using the license.

Note on cores and licenses
The licensing of Altova server products is based on the number of processor cores available on the product machine. For example, a dual-core processor has two cores, a quad-core processor four cores, a hexa-core processor six cores, and so on. The number of cores licensed for a product on a particular server machine must be greater than or equal to the number of cores available on that server, whether it's a physical or virtual machine.

For example, if a server has eight cores (an octa-core processor), you must purchase at least an 8-core license. You can also combine licenses to achieve the core count. So, two 4-core

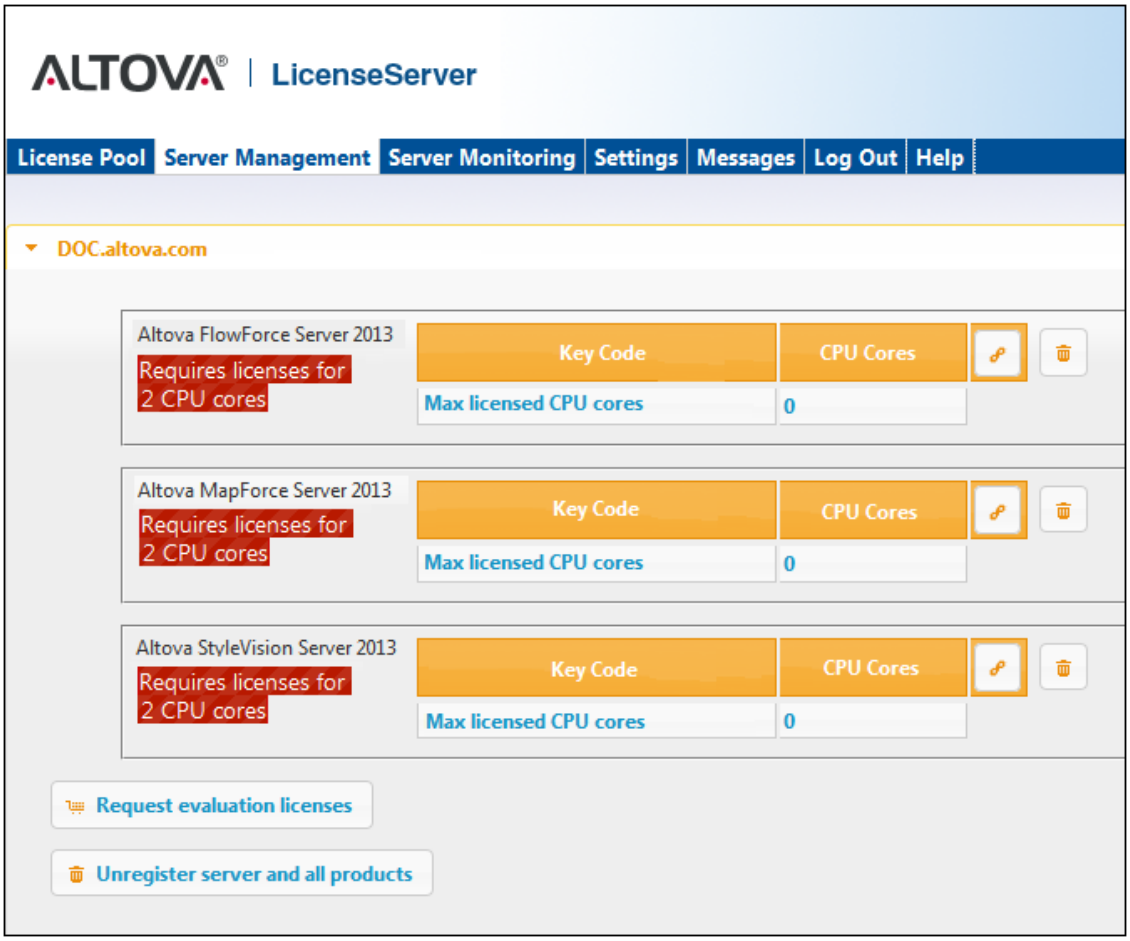
licenses can also be used for an octa-core server instead of an 8-core license.

If you are using a computer server with a large number of CPU cores but only have a low volume to process, you may also create a virtual machine that is allocated a smaller number of cores, and purchase a license for that number. Such a deployment, of course, will have less processing speed than if all available cores on the server were utilized.

Note: Each license can be used for only one client machine at a time, even if it has unused licensing capacity. For example, if a 10-core license is used for a client machine that has 6 CPU cores, then the remaining 4 cores of the license cannot be used simultaneously for another client machine.

10.7.2 Server Management

In the **Server Management** tab (*screenshot below*), you can assign licenses to registered products.



Note the following points about the Server Management tab:

- Each product is listed under the name of its client machine. In the screenshot above, one client machine, named `Doc.altova.com`, has three Altova products registered with the LicenseServer. If an Altova product on a different client machine is registered with this LicenseServer, then that client machine, with its registered products, will also be listed in the Server Management tab.
- Each registered Altova product on a client machine has its own *Key Code* entry, which takes the key code of a license. A registered product's key code is assigned by clicking its **Edit Assigned Licenses** button and selecting the required license from those available for that product (for example, FlowForce Server) in the license pool. This procedure is explained in more detail below.
- Each product also has a line stating how many CPU cores need to be licensed to run that product on that client. If the number of licensed cores is less than the number required, then the information is marked in red (*see screenshot above*). (The number of CPU cores that need to be licensed is the number of CPU cores on that client and is obtained from the client machine by LicenseServer.)

Icons in the Server Management tab

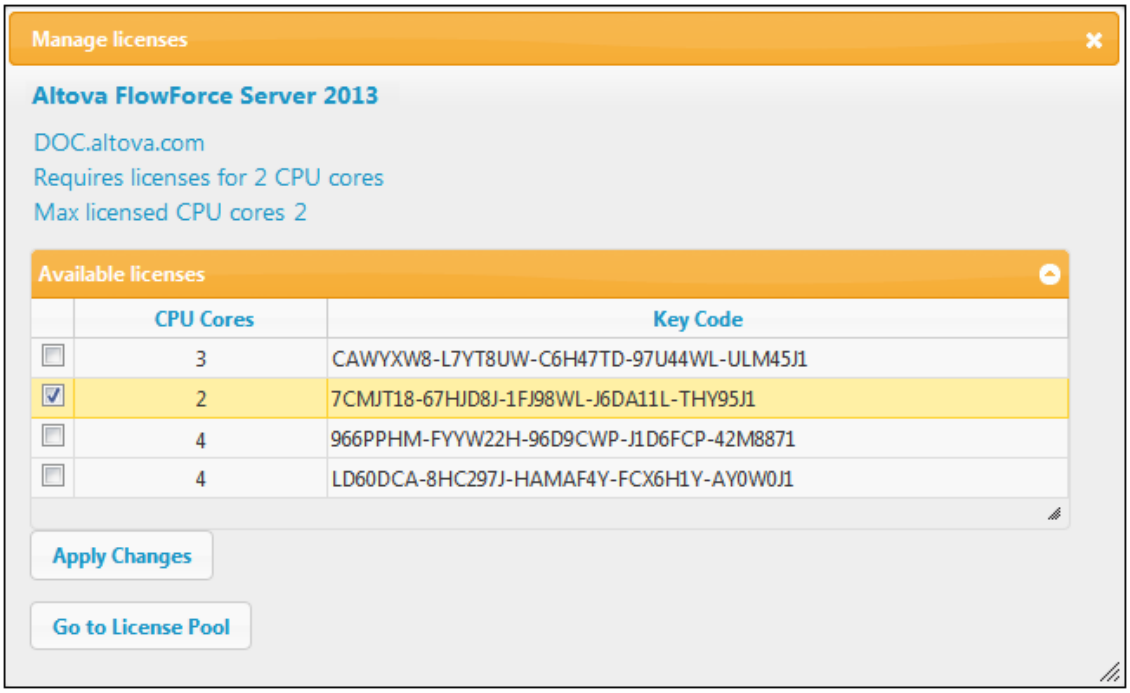
Edit Assigned Licenses. Available with each product. Pops up the Manage Licenses dialog, in which new licenses can be assigned to the product and already assigned licenses can be edited.

Show Licenses. Appears with each license. Switches to the License Pool tab and highlights the selected license, so that license details can be read.

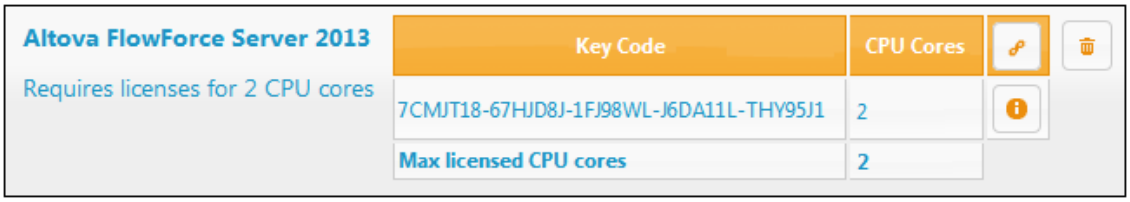
Unregister This Product. Available with each product. The selected product (on the selected client machine) will be unregistered from LicenseServer.

Assigning a license

To assign a license to a registered product, click the **Edit Assigned Licenses** button of that product. This pops up the Manage Licenses dialog (*screenshot below*).



Select the license you wish to assign. After selecting the license/s, click **Apply Changes**. The license/s will be assigned to that product and displayed in the Server Management tab (*see screenshot below*).



Requesting an evaluation license

You can obtain a 30-day free evaluation license for each of a client's installed Altova products

that have been registered with LicenseServer. Click the **Request Evaluation Licenses** button near the bottom of the Server Management tab. A dialog pops up containing a list of the Altova server products (on that client machine) which have been registered with LicenseServer. Make sure that the products for which you want an evaluation license are checked, then fill in the registration fields, and send the request. You will receive an e-mail from Altova containing the 30-day evaluation license/s. The number of cores for which the license will be valid per product will be exactly the number required by the product at the time the request is sent. Save the license/s to disk and upload to the license pool.

Unregistering products

Each Altova product registered with LicenseServer is listed in the Server Management tab under its client machine name and has an **Unregister** icon to its right. Click this icon to unregister the product. If a license was assigned to the product, the assignment will be terminated when the product is unregistered. To unregister all products, click the **Unregister Server and All Products** button at the bottom of the Server Management tab (*see first screenshot in this section*).

To re-register a product, go to the product's Setup page.

For more information, see the section, Assigning Licenses to Registered Products.

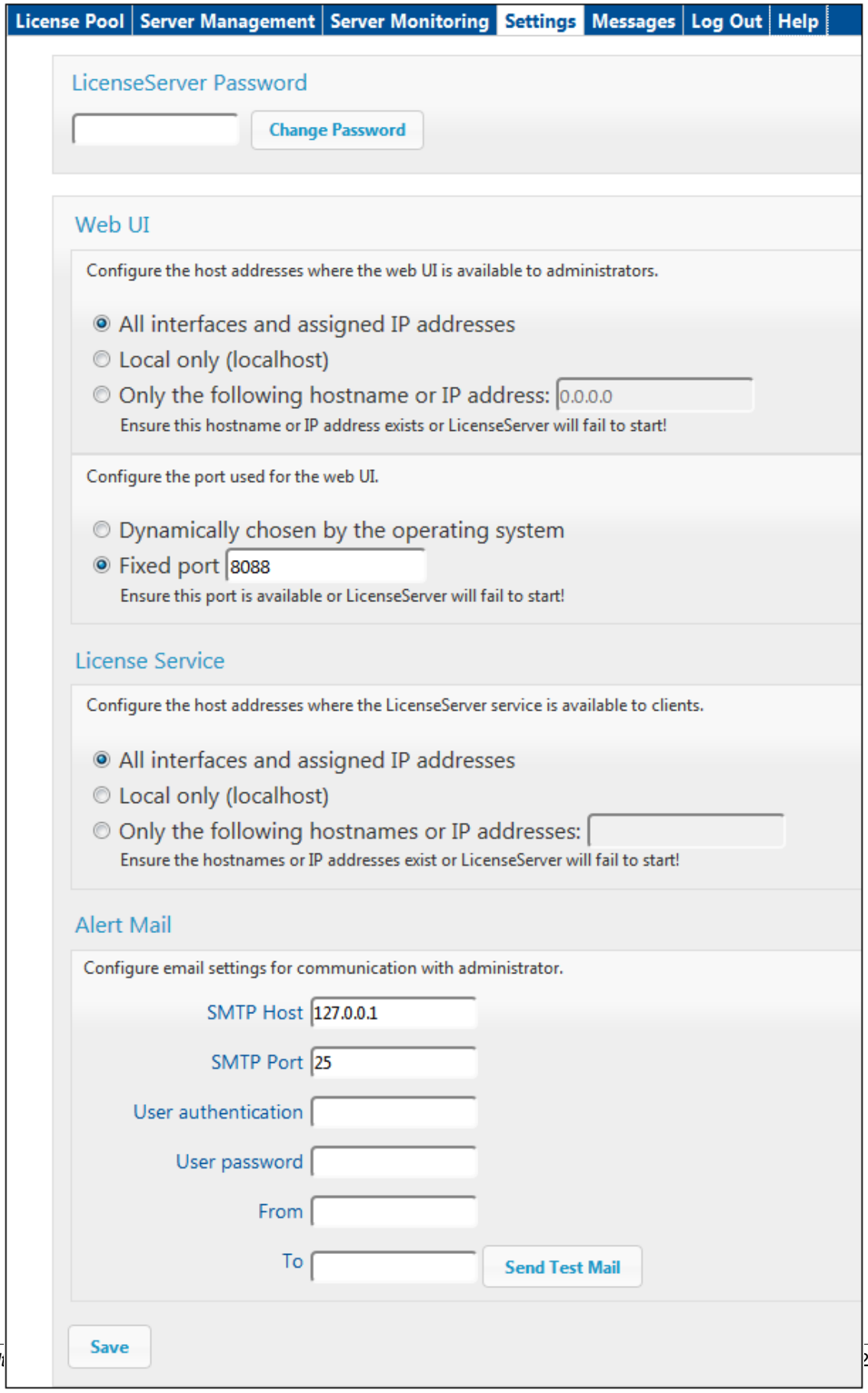
10.7.3 Server Monitoring

The **Server Monitoring** tab provides an overview of servers currently running licensed Altova products. It contains product information along with information about users and licenses.

10.7.4 Settings

The **Settings** tab is as shown below. You can set the following:

- The password for logging in to LicenseServer.
- Network settings for the web-based configuration page (Web UI), and for LicenseServer (License Service). These settings are described below the screenshot.
- Email server settings and the alert mail recipient. These are described below the screenshot.


License Pool
Server Management
Server Monitoring
Settings
Messages
Log Out
Help
LicenseServer Password
Change Password
Web UI
Configure the host addresses where the web UI is available to administrators.
All interfaces and assigned IP addresses
Local only (localhost)
Only the following hostname or IP address:
0.0.0.0
Ensure this hostname or IP address exists or LicenseServer will fail to start!
Configure the port used for the web UI.
Dynamically chosen by the operating system
Fixed port
8088
Ensure this port is available or LicenseServer will fail to start!
License Service
Configure the host addresses where the LicenseServer service is available to clients.
All interfaces and assigned IP addresses
Local only (localhost)
Only the following hostnames or IP addresses:
Ensure the hostnames or IP addresses exist or LicenseServer will fail to start!
Alert Mail
Configure email settings for communication with administrator.
SMTP Host
127.0.0.1
SMTP Port
25
User authentication
User password
From
To
Send Test Mail
Save

Network settings
Administrators can specify network access points to the LicenseServer configuration page and to LicenseServer:

- *Web UI:* Allowed IP addresses can vary from all interfaces and IP addresses on that machine to a fixed address, and ports can be either dynamically calculated or fixed. This allows a wide range of allowed IP-Address:Port settings. The default port setting is **8088**.
- *License Service:* IP addresses can vary from all interfaces and IP addresses on that machine to a fixed address. The port number is fixed at **35355**.

By default, these settings allow unrestricted access to LicenseServer and its configuration page from within the networks to which LicenseServer is connected. If you wish to restrict access to either LicenseServer or its configuration page, enter the appropriate settings and click **Save**.

Alert Mail settings
Altova LicenseServer needs to be connected to the `altova.com` server. If the connection is broken for more than 24*5 hours (5 days), LicenseServer will not allow licenses. As a result, work sessions with Altova products licensed by LicenseServer could be disrupted. In order to alert that administrator that a connection is broken, an alert mail can be sent to an email address. The Alert Mail pane of this page enables settings to be made for sending alert mails to an administrator's email address.

SMTP Host and *SMTP Port* are the access details of the email server from which the email alert will be sent. *User Authentication* and *User Password* are the user's credentials for accessing the email server. The *From* field takes the address of the email account from which the email will be sent. The *To* field takes the recipient's email address.

Click **Save** when done. After saving the Alert Mail settings, email alerts will be sent to the address specified whenever a significant event occurs, such as when connection to `altova.com` is lost. These events are also recorded in the Messages tab.

10.7.5 Messages, Log Out

The **Messages** tab displays all messages relevant to licenses in the license pool of the LicenseServer. Each message has a **Delete** button that allows you to delete that particular message.

The **Log Out** tab serves as the Log Out button. Clicking the tab logs you out immediately and then displays the Login mask.

Index

I

J

L

M

N

P

R

S

U

V

W

X