Altova RaptorXML+XBRL Server 2015

User and Reference Manual

Altova RaptorXML+XBRL Server 2015 User & Reference Manual

Published: 2014

Table of Contents

Chapter 1

About RaptorXML+XBRL Server

1 About RaptorXML+XBRL Server

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.

Editions and operating systems
There are two editions of RaptorXML, each suitable for a different set of requirements. These editions are described in the section Editions and Interfaces. RaptorXML is 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 XPath/XQuery Functions
- Altova LicenseServer

Last updated: 09-15-2014

1.1 Editions and Interfaces

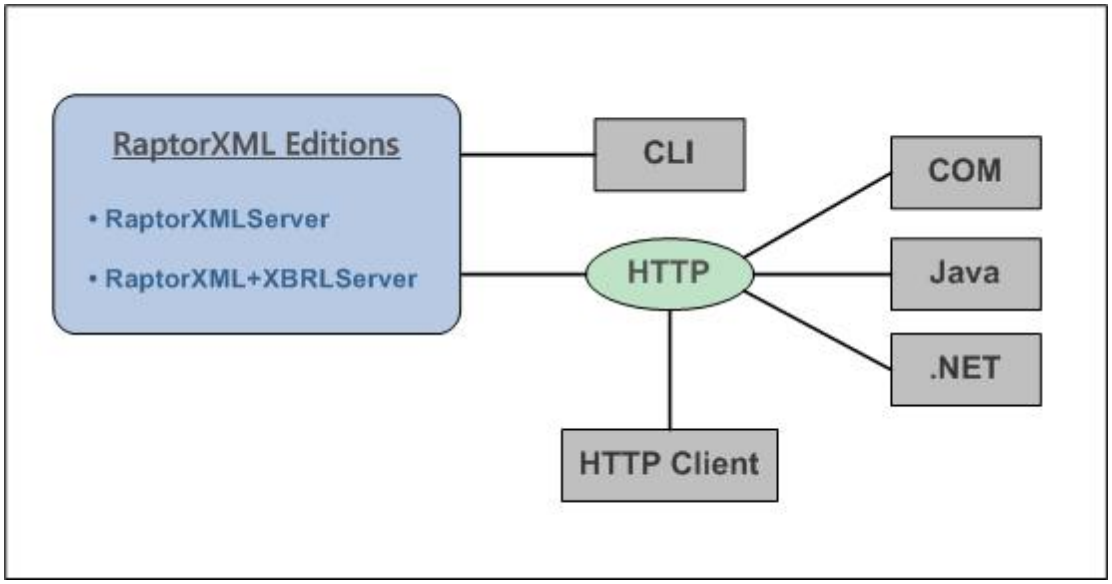
RaptorXML is available in the following editions:

- *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.

Interfaces
RaptorXML is accessed via the following interfaces:

- A command line interface (CLI)
- A COM interface on Windows systems
- A .NET interface on Windows systems
- A Java interface on Windows, Linux, and MacOS systems
- An HTTP interface that can be accessed by an HTTP client
- 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

The diagram below shows how RaptorXML is accessed via its 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+XBRL Server is supported on the following operating systems:

- *Windows*
  Windows XP (SP2 for x64; SP3 for x86), Windows Vista, Windows 7, Windows 8, or newer

- *Windows Server*
  Windows Server 2008 R2 or newer

- *Linux*
  - CentOS 6 or newer
  - RedHat 6 or newer
  - Debian 6 or newer
  - Ubuntu 12.04 or newer

  Note that the Qt library (version 4 or later), available under GNU GPL and LGPL, must be installed.

- *Mac OS X*
  Mac OS X 10.7 or newer

RaptorXML is 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. 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, and the Table Linkbase 1.0 proposed recommendation of 18 December 2013.

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 Specifications 1.0
  - Aspect Cover Filters

  - Boolean Filters
  - Concept Filters
  - Concept Relation Filters
  - Consistency Assertions
  - Custom Function Implementation
  - Dimension Filters
  - Entity Filters
  - Existence Assertions
  - Formula
  - Function Registry
  - 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
- Table Linkbase 1.0 (Proposed Recommendation of 18 December 2013)
- Function Definitions
- Generic Links 1.0
  - General Filters
  - Generic Labels

Chapter 2

Setting Up RaptorXML

2 Setting Up RaptorXML

This section describes procedures for 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 Setup on Windows

This section describes the installation and licensing of RaptorXML+XBRL Server on Windows systems.

Installation on Windows

- System requirements
- Installing RaptorXML+XBRL Server
- Altova LicenseServer
- LicenseServer versions
- Trial license
- Application folder location

Licensing on Windows

- Start ServiceController
- Start LicenseServer
- Start RaptorXML+XBRL Server
- Register RaptorXML+XBRL Server
- Assign a license

2.1.1 Installation on Windows

RaptorXML+XBRL Server is available for installation on Windows systems. Its installation and setup procedure is described below.

- System requirements
  - *Windows*
    Windows XP (SP2 for x64; SP3 for x86), Windows Vista, Windows 7, Windows 8, or newer
  - *Windows Server*
    Windows Server 2008 R2 or newer

- Installing RaptorXML+XBRL Server
  RaptorXML+XBRL Server can be installed on Windows systems as follows:

  - As a separate standalone server product called RaptorXML+XBRL Server. To install RaptorXML+XBRL Server, download and run the RaptorXML+XBRL Server installer. Follow the onscreen instructions.
  - As part of the FlowForce Server installation package. To install RaptorXML+XBRL Server as part of the FlowForce Server package, download and run the FlowForce Server installer. Follow the onscreen instructions and make sure you check the option for installing RaptorXML+XBRL Server.

  The installers of both RaptorXML+XBRL Server and FlowForce Server are available at the Altova website and will install the products with the necessary registrations. After installation, the RaptorXML+XBRL Server executable will be located by default at:

  ```
  <ProgramFilesFolder>\Altova\RaptorXMLXBRLServer2015\bin\RaptorXMLXBRL.exe
  ```

  All the necessary registrations to use RaptorXML+XBRL Server via a COM interface, as a Java interface, and in the .NET environment will be done by the installer. This includes registering the RaptorXML+XBRL Server 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.

- Altova LicenseServer
  - In order for RaptorXML+XBRL Server to work, it must be licensed via an Altova LicenseServer on your network.
  - When you install RaptorXML+XBRL Server or FlowForce Server on Windows systems, an option is available that allows you to download and install Altova LicenseServer together with RaptorXML+XBRL Server or FlowForce Server.
  - If an Altova LicenseServer is already installed on your network, you do not need to install another one—unless a newer version of Altova LicenseServer is required. (*See next point, LicenseServer versions*.)
  - During the installation process of RaptorXML+XBRL Server or FlowForce Server, check or uncheck the option for installing Altova LicenseServer as appropriate.

  See the section, Licensing on Windows, for more information about how to register and license RaptorXML+XBRL Server with Altova LicenseServer.

- LicenseServer versions

- Altova server products must be licensed either with the version of LicenseServer that is appropriate to the installed RaptorXML+XBRL Server version, or with a later version of LicenseServer.
- The LicenseServer version that is appropriate for a particular version of RaptorXML +XBRL Server is displayed during the installation of RaptorXML+XBRL Server. You can install this version of LicenseServer along with RaptorXML+XBRL Server, or you can install LicenseServer separately.
- Before installing a newer version of LicenseServer, any older one must be de-installed. The LicenseServer installer will do this automatically if it detects an older version.
- LicenseServer versions are backwards compatible. They will work with older versions of RaptorXML+XBRL Server.
- If you install a new version of RaptorXML+XBRL Server and if your installed LicenseServer version is older than the appropriate LicenseServer, install the latest version available on the Altova website.
- At the time of LicenseServer de-installation, all registration and licensing information held in the older version of LicenseServer will be saved to a database on your server machine. This data will be imported automatically into the newer version when the newer version is installed.
- The version number of the currently installed LicenseServer is given at the bottom of the LicenseServer configuration page (all tabs).

Current version: 1.11

▼ Trial license

During the installation process, you will be given the option of requesting a 30-day trial license for RaptorXML+XBRL Server. After submitting the request, a trial license will be sent to the email address you registered.

▼ Application folder location

The application will be installed in the following folder:

Windows XP	`C:\Program Files\Altova\`
Windows Vista, Windows 7/8	`C:\Program Files\Altova\`
32 bit Version on 64-bit OS	`C:\Program Files (x86)\Altova\`

2.1.2 Licensing on Windows

RaptorXML+XBRL Server must be licensed with an Altova LicenseServer in order to run it. Licensing is a two-step process:

1. **Register RaptorXML+XBRL Server** with LicenseServer. Registration is done from RaptorXML+XBRL Server.
2. **Assign a license** to RaptorXML+XBRL Server. License-assigning is done from LicenseServer.

The steps you need to carry out are given below.

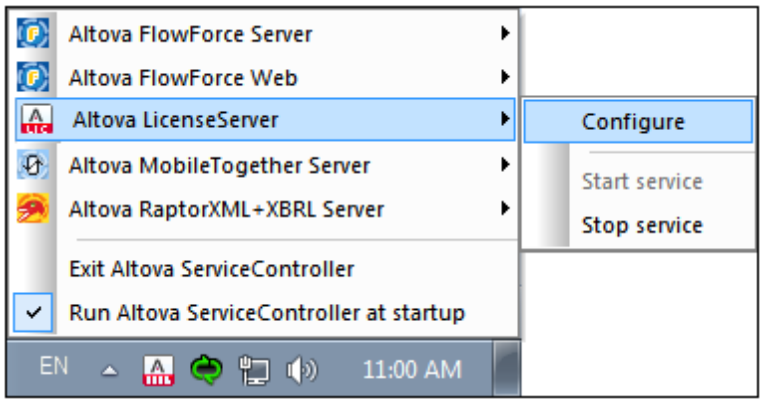
▼ Start ServiceController

Altova ServiceController is started in order to start Altova LicenseServer and Altova RaptorXML +XBRL Server.

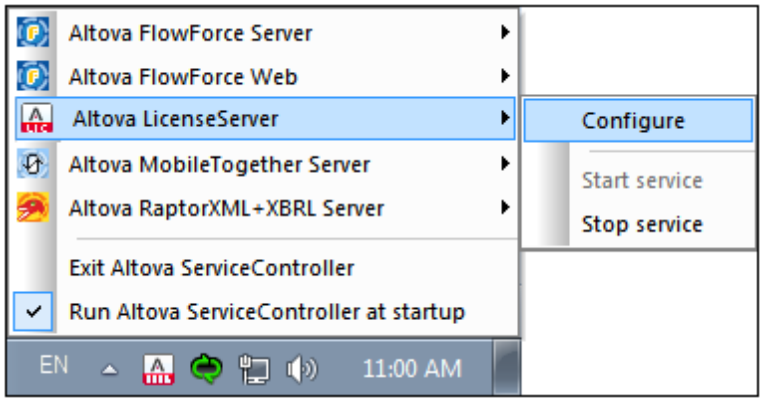
Altova ServiceController (ServiceController for short) is an application for conveniently starting, stopping and configuring Altova services **on Windows systems**.

ServiceController is installed with Altova LicenseServer and with *Altova server products that are installed as services* (FlowForce Server, RaptorXML(+XBRL) Server, and Mobile Together Server). It can be started by clicking **Start | Altova LicenseServer | Altova ServiceController**. (This command is also available in the **Start** menu folders of *Altova server products that are installed as services* (FlowForce Server, RaptorXML(+XBRL) Server, and Mobile Together Server).) After ServiceController has been started, it can be accessed via the system tray (*screenshot below*).

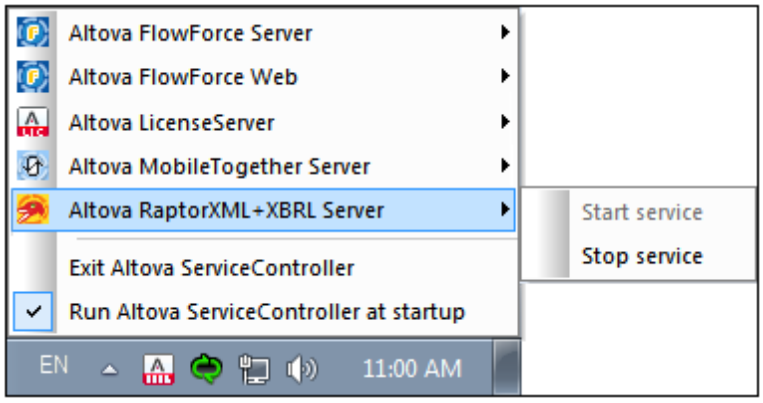
To specify that ServiceController starts automatically on logging in to the system, click the **ServiceController** icon in the system tray to display the **ServiceController** menu (*screenshot below*), and then toggle on the command **Run Altova ServiceController at Startup**. (This command is toggled on by default.) To exit ServiceController, click the **ServiceController** icon in the system tray and, in the menu that appears (*see screenshot below*), click **Exit Altova ServiceController**.



- Start LicenseServer
  To start LicenseServer, click the **ServiceController** icon in the system tray, hover 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.



- Start RaptorXML+XBRL Server
  To start RaptorXML+XBRL Server, click the **ServiceController** icon in the system tray, hover over **Altova RaptorXML+XBRL Server** in the menu that pops up (*see screenshot below*), and then select **Start Service** from the RaptorXML+XBRL Server submenu. If RaptorXML+XBRL Server is already running, the *Start Service* option will be disabled.



- Register RaptorXML+XBRL Server

  - *Register RaptorXML+XBRL Server through FlowForce Server*
    If RaptorXML+XBRL Server was installed as part of a FlowForce Server installation, registering FlowForce Server with LicenseServer will automatically also register RaptorXML+XBRL Server. How to register FlowForce Server is described in the FlowForce Server documentation. Essentially: (i) Start Altova FlowForce Web as a service via ServiceController (*see previous point*); (ii) Enter your password to access the Setup page; (iii) Select the LicenseServer name or address and click **Register with LicenseServer**.

After successful registration, go to the Server Management tab of LicenseServer's configuration page to assign a license to RaptorXML+XBRL Server.

- *Register a standalone RaptorXML+XBRL Server*
  Register RaptorXML+XBRL Server via:

  - its CLI, using the `licenseserver` command:
    `RaptorXMLXBRL licenseserver [options] ServerName-Or-IP-Address`

    For example, if `localhost` is the name of the server on which LicenseServer is installed:
    `RaptorXMLXBRL licenseserver localhost`

  After successful registration, go to the Server Management tab of LicenseServer's configuration page to assign a license to RaptorXML+XBRL Server.

- Assign a license
  After successfully registering RaptorXML+XBRL Server, it will be listed in the Server Management tab of the configuration page of LicenseServer. Go there and assign a license to RaptorXML+XBRL Server.

  Note on cores and licenses
  The licensing of Altova server products, *except MobileTogether Server****, 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.

  *** MobileTogether Server licenses are assigned on the basis of the number of users, that is, the number of client devices that connect to MobileTogether Server.

2.2 Setup on Linux

This section describes the installation and licensing of RaptorXML+XBRL Server on Linux systems (Debian, Ubuntu, CentOS, RedHat).

Installation on Linux

- System requirements
- Note about root user
- Uninstall old versions of Altova server products
- Download the Linux package
- Install RaptorXML+XBRL Server
- Altova LicenseServer
- LicenseServer versions
- Trial license

Licensing on Linux

- Note about root user
- Start LicenseServer
- Start RaptorXML+XBRL Server
- Register RaptorXML+XBRL Server
- Assign a license

2.2.1 Installation on Linux

RaptorXML+XBRL Server is available for installation on Linux systems. Its installation and setup procedure is described below.

- System requirements
  - *Linux*
    - CentOS 6 or newer
    - RedHat 6 or newer
    - Debian 6 or newer
    - Ubuntu 12.04 or newer

    Note that the Qt library (version 4 or later), available under GNU GPL and LGPL, must be installed.

- FlowForce Server integration
  If you are installing RaptorXML+XBRL Server together with FlowForce Server, it is recommended to install FlowForce Server first. Otherwise, after having installed both RaptorXML+XBRL Server and FlowForce Server, run the following command:

```
cp /opt/Altova/RaptorXMLXBRLServer2015/etc/*.tool /opt/Altova/
FlowForceServer2015/tools
```

  This command copies the **.tool** file from **/etc** directory of RaptorXML+XBRL Server to the FlowForce Server **/tools** directory. The **.tool** file is required by FlowForce Server; it contains the path to the RaptorXML+XBRL Server executable. You do not need to run this command if you install FlowForce Server before installing RaptorXML+XBRL Server.

- Note about root user
  You must have administrator (root) privileges to be able to install RaptorXML+XBRL Server. Installation must be done, therefore, as the root user. If you are logged in as `root`, you can leave out the `sudo` keyword from the commands listed below.

- Uninstall old versions of Altova server products
  If you need to uninstall a previous version, do this as follows. On the Linux command line interface (CLI), you can check which Altova server products are 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 below in *Installing RaptorXML+XBRL Server*.

  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
```

- Download the Linux package

  RaptorXML+XBRL Server installation packages for the following Linux systems are available on the Altova website.

Distribution	Package extension
Debian 6 and higher	.deb
Ubuntu12.04 and higher	.deb
CentOS 6 and higher	.rpm
RedHat 6 and higher	.rpm

  After downloading the Linux package, copy it 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.

- Install 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-2015-debian.deb
[Ubuntu]:   sudo dpkg --install raptorxmlxbrlserver-2015-ubuntu.deb
[CentOS]:   sudo rpm -ivh raptorxmlxbrlserver-2015-1.x86_64.rpm
[RedHat]:   sudo rpm -ivh raptorxmlxbrlserver-2015-1.x86_64.rpm
```

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

```
/opt/Altova/RaptorXMLXBRLServer2015
```

- Altova LicenseServer

  In order for any Altova Server product—including RaptorXML+XBRL Server—to run, that server product must be licensed via an Altova LicenseServer on your network.

  On Linux systems, Altova LicenseServer will need to be installed separately. Download LicenseServer from the Altova website and copy the package to any directory on the Linux system. Install it just like you installed RaptorXML+XBRL Server (*see previous step*).

```
[Debian]:   sudo dpkg --install licenseserver-1.11-debian.deb
[Ubuntu]:   sudo dpkg --install licenseserver-1.11-ubuntu.deb
[CentOS]:   sudo rpm -ivh licenseserver-1.11-1.x86_64.rpm
[RedHat]:   sudo rpm -ivh licenseserver-1.11-1.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 on Linux.

- LicenseServer versions
  - Altova server products must be licensed either with the version of LicenseServer that is appropriate to the installed RaptorXML+XBRL Server version, or with a later version of LicenseServer.
  - The LicenseServer version that is appropriate for a particular version of RaptorXML +XBRL Server is displayed during the installation of RaptorXML+XBRL Server. You can install this version of LicenseServer along with RaptorXML+XBRL Server, or you can install LicenseServer separately.
  - Before installing a newer version of LicenseServer, any older one must be de-installed. The LicenseServer installer will do this automatically if it detects an older version.
  - LicenseServer versions are backwards compatible. They will work with older versions of RaptorXML+XBRL Server.
  - If you install a new version of RaptorXML+XBRL Server and if your installed LicenseServer version is older than the appropriate LicenseServer, install the latest version available on the Altova website.
  - At the time of LicenseServer de-installation, all registration and licensing information held in the older version of LicenseServer will be saved to a database on your server machine. This data will be imported automatically into the newer version when the newer version is installed.
  - The version number of the currently installed LicenseServer is given at the bottom of the LicenseServer configuration page (all tabs).

    Current version: 1.11

- Trial license
  During the installation process, you will be given the option of requesting a 30-day trial license for RaptorXML+XBRL Server. After submitting the request, a trial license will be sent to the email address you registered.

2.2.2 Licensing on Linux

RaptorXML+XBRL Server must be licensed with an Altova LicenseServer in order to run it. Licensing is a two-step process:

1. **Register RaptorXML+XBRL Server** with LicenseServer. Registration is done from RaptorXML+XBRL Server.
2. **Assign a license** to RaptorXML+XBRL Server. License-assigning is done from LicenseServer.

The steps you need to carry out are given below.

▼ Note about root user

You must have administrator (root) privileges to be able to install RaptorXML+XBRL Server. Installation must be done, therefore, as the root user. If you are logged in as `root`, you can leave out the `sudo` keyword from the commands listed below.

▼ Start LicenseServer

To correctly register and license RaptorXML+XBRL Server with LicenseServer, LicenseServer must be running as a daemon on the network. Start LicenseServer as a daemon 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, replace **`start`** with **`stop`** in the above command. For example:

```
sudo /etc/init.d/licenseserver stop
```

▼ Start RaptorXML+XBRL Server

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

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

▼ Register RaptorXML+XBRL Server

Register RaptorXML+XBRL Server via:

- its CLI, using the `licenseserver` command:
  `sudo /opt/Altova/RaptorXMLXBRLServer2015/bin/raptorxmlxbrl licenseserver [options] ServerName-Or-IP-Address`

  For example, if `localhost` is the name of the server on which LicenseServer is installed:

```
sudo /opt/Altova/RaptorXMLXBRLServer2015/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:

```
/opt/Altova/RaptorXMLXBRLServer2015/bin/
```

After successful registration, go to the Server Management tab of LicenseServer's configuration page to assign a license to RaptorXML+XBRL Server.

▼ Assign a license

After successfully registering RaptorXML+XBRL Server, it will be listed in the Server Management tab of the configuration page of LicenseServer. Go there and assign a license to RaptorXML+XBRL Server.

Note on cores and licenses

The licensing of Altova server products, *except MobileTogether Server****, 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.

*** MobileTogether Server licenses are assigned on the basis of the number of users, that is, the number of client devices that connect to MobileTogether Server.

2.3 Setup on Mac OS X

This section describes the installation and licensing of RaptorXML+XBRL Server on Mac OS X systems.

Installation on Mac OS X

- System requirements
- Note about root user
- Uninstall old versions of Altova server products
- Download the Mac OS X package
- Install RaptorXML+XBRL Server
- Altova LicenseServer
- LicenseServer versions
- Trial license

Licensing on Mac OS X

- Note about root user
- Start LicenseServer
- Start RaptorXML+XBRL Server
- Register RaptorXML+XBRL Server
- Assign a license

2.3.1 Installation on Mac OS X

RaptorXML+XBRL Server is available for installation on Linux systems. Its installation and setup procedure is described below.

- System requirements
  - *Mac OS X*
    Mac OS X 10.7 or newer

- FlowForce Server integration
  If you are installing RaptorXML+XBRL Server together with FlowForce Server, it is recommended to install FlowForce Server first. Otherwise, after having installed both RaptorXML+XBRL Server and FlowForce Server, run the following command:

```
cp /usr/local/Altova/RaptorXMLXBRLServer2015/etc/*.tool /usr/local/Altova/
FlowForceServer2015/tools
```

  This command copies the **.tool** file from **/etc** directory of RaptorXML+XBRL Server to the FlowForce Server **/tools** directory. The **.tool** file is required by FlowForce Server; it contains the path to the RaptorXML+XBRL Server executable. You do not need to run this command if you install FlowForce Server before installing RaptorXML+XBRL Server.

- Note about root user
  You must have administrator (root) privileges to be able to install RaptorXML+XBRL Server. Installation must be done, therefore, as the root user. If you are logged in as `root`, you can leave out the `sudo` keyword from the commands listed below.

- Uninstall old versions of Altova server products
  Before uninstalling RaptorXML+XBRL Server, stop the service with the following command:

```
sudo launchctl unload /Library/LaunchDaemons/
com.altova.RaptorXMLXBRLServer2015.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/RaptorXMLXBRLServer2015/
```

  If you need to uninstall an old version of Altova LicenseServer, you must first stop it running as a service. Do this with the following command:

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

  To check whether the service has been stopped, open the Activity Monitor terminal and make sure that LicenseServer is not in the list. Then proceed to uninstall in the same way as described above for RaptorXML+XBRL Server.

- Download the Mac OS X package
  After downloading the MacOS X package from the Altova website, copy the package to any directory on the Mac OS X system. Since you will need to have an Altova LicenseServer installed 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.

- Install 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/RaptorXMLXBRLServer2015`

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

- Altova LicenseServer
  In order for any Altova Server product—including RaptorXML+XBRL Server—to run, that server product 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 onscreen 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 on Mac OS X.

- LicenseServer versions
  - Altova server products must be licensed either with the version of LicenseServer that is appropriate to the installed RaptorXML+XBRL Server version, or with a later version of LicenseServer.
  - The LicenseServer version that is appropriate for a particular version of RaptorXML +XBRL Server is displayed during the installation of RaptorXML+XBRL Server. You can install this version of LicenseServer along with RaptorXML+XBRL Server, or you can install LicenseServer separately.
  - Before installing a newer version of LicenseServer, any older one must be de-installed. The LicenseServer installer will do this automatically if it detects an older version.
  - LicenseServer versions are backwards compatible. They will work with older versions of RaptorXML+XBRL Server.
  - If you install a new version of RaptorXML+XBRL Server and if your installed LicenseServer version is older than the appropriate LicenseServer, install the latest version available on the Altova website.
  - At the time of LicenseServer de-installation, all registration and licensing information

held in the older version of LicenseServer will be saved to a database on your server machine. This data will be imported automatically into the newer version when the newer version is installed.

- The version number of the currently installed LicenseServer is given at the bottom of the LicenseServer configuration page (all tabs).

  Current version: 1.11

▼ Trial license

During the installation process, you will be given the option of requesting a 30-day trial license for RaptorXML+XBRL Server. After submitting the request, a trial license will be sent to the email address you registered.

2.3.2 Licensing on Mac OS X

RaptorXML+XBRL Server must be licensed with an Altova LicenseServer in order to run it. Licensing is a two-step process:

1. **Register RaptorXML+XBRL Server** with LicenseServer. Registration is done from RaptorXML+XBRL Server.
2. **Assign a license** to RaptorXML+XBRL Server. License-assigning is done from LicenseServer.

The steps you need to carry out are given below.

▼ Note about root user

You must have administrator (root) privileges to be able to install RaptorXML+XBRL Server. Installation must be done, therefore, as the root user. If you are logged in as `root`, you can leave out the `sudo` keyword from the commands listed below.

▼ Start LicenseServer

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

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

If at any time you need to stop LicenseServer, replace **load** with **unload** in the above command:

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

▼ Start RaptorXML+XBRL Server

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

```
sudo launchctl load /Library/LaunchDaemons/
com.altova.RaptorXMLXBRLServer2015.plist
```

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

```
sudo launchctl unload /Library/LaunchDaemons/
com.altova.RaptorXMLXBRLServer2015.plist
```

▼ Register RaptorXML+XBRL Server

Register RaptorXML+XBRL Server via:

- its CLI, using the licenseserver command:

  ```
  sudo /usr/local/Altova/RaptorXMLXBRLServer2015/bin/RaptorXMLXBRL
  licenseserver [options] ServerName-Or-IP-Address
  ```

  For example, if `localhost` is the name of the server on which LicenseServer is installed:

  ```
  sudo /usr/local/Altova/RaptorXMLXBRLServer2015/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/RaptorXMLXBRLServer2015/bin/
```

After successful registration, go to the Server Management tab of LicenseServer's configuration page to assign a license to RaptorXML+XBRL Server.

- Assign a license

  After successfully registering RaptorXML+XBRL Server, it will be listed in the Server Management tab of the configuration page of LicenseServer. Go there and assign a license to RaptorXML+XBRL Server.

  Note on cores and licenses

  The licensing of Altova server products, *except MobileTogether Server****, 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.

  *** MobileTogether Server licenses are assigned on the basis of the number of users, that is, the number of client devices that connect to MobileTogether Server.

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:

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\RaptorXMLXBRLServer2015\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\RaptorXMLXBRLServer2015\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\Common2015
%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.

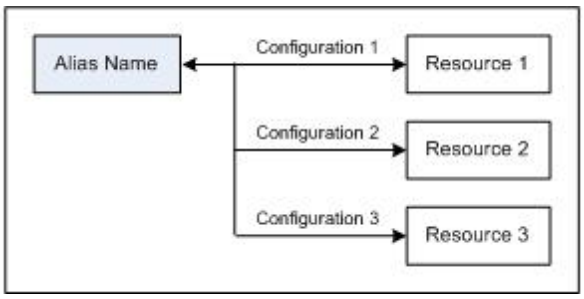
%LocalAppDataFolder%	Full path to the file system directory that serves as the data repository for local (non-roaming) applications.
%MyPicturesFolder%	Full path to the MyPictures folder.

2.5 Global Resources

This section:

- About global resources
- Using global resources

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



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

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

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

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

For example, if you wish to transform `input.xml` with `transform.xslt` to `output.html`, this

would typically be achieved on the CLI with the following command that uses filenames:

```
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 RaptorXML+XBRL 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 RaptorXML+XBRL 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 in a command via HTTP to RaptorXML+XBRL 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+XBRL Server executable for use with the command line interface (CLI) is located by default at:

Windows	<*ProgramFilesFolder*>\Altova\RaptorXMLXBRLServer2015\bin**RaptorXMLXBRL.exe**
Linux	\opt\Altova\RaptorXMLXBRLServer2015\bin**raptorxmlxbrl**
Mac	\usr\local\Altova\RaptorXMLXBRLServer2015\bin**raptorxmlxbrl**

▼ Casing on the command line

RaptorXMLXBRL *on Windows*
raptorxmlxbrl *on Unix (Linux, Mac)*

* Note that lowercase (raptorxmlxbrl) works on all platforms (Windows, Linux, and Mac), while upper-lower (RaptorXMLXBRL) works only on Windows and Mac.

Usage
The command line syntax is:

Windows	**RaptorXMLXBRL --h | --help | --version | <command> [options] [arguments]**
Linux	**raptorxmlxbrl --h | --help | --version | <command> [options] [arguments]**
Mac	**raptorxmlxbrl --h | --help | --version | <command> [options] [arguments]**

RaptorXMLXBRL	Calls the application on Windows platforms.
raptorxmlxbrl	Calls the application on Unix platforms (Linux and Mac).
--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.

Windows	**RaptorXMLXBRL valxml-withdtd | xml [options] *InputFile***
Linux	**raptorxmlxbrl valxml-withdtd | xml [options] *InputFile***
Mac	**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

▼ Casing on the command line

RaptorXMLXBRL *on Windows*
raptorxmlxbrl *on Unix (Linux, Mac)*

* Note that lowercase (raptorxmlxbrl) works on all platforms (Windows, Linux, and Mac), while upper-lower (RaptorXMLXBRL) works only on Windows and Mac.

Options
The command's options are listed below, organized into groups. Values 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.

▼ Validation and processing

▼ **dtd**

--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.

▼ **listfile**

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

namespaces

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

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

streaming

`--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 that the `--streaming` option is ignored if `--parallel-assessment` is set to `true`.
Note: Boolean option values are set to `true` if the option is specified without a value.

Catalogs and global resources

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 (`<installation-folder>\Altova\RaptorXMLXBRLServer2015\etc\RootCatalog.xml`). See the section, XML Catalogs, for information about working with catalogs.

user-catalog

`--user-catalog =` *`FILE`*
Specifies the absolute path to an XML catalog to be used in addition to the root catalog. See the section, XML Catalogs, for information about working with catalogs.

enable-globalresources

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

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

globalresourceconfig [gc]

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

globalresourcefile [gr]

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

Messages, errors, help, timeout, version

error-format

--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.

error-limit

--error-limit = *N*
Specifies the error limit. Default value is `100`. Values of `1` to `999` are allowed. Useful for limiting processor use during validation. When the error limit is reached, validation stops.

help

--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`.)

log-output

--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.

network-timeout

--network-timeout = *VALUE*
Specifies the timeout in seconds for remote I/O operations. Default is: `40`.

verbose

--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.

verbose-output

--verbose-output = *FILE*
Writes verbose output to *FILE*.

version

--version

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

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.

Windows	**RaptorXMLXBRL valxml-withxsd | xsi [options] *InputFile***
Linux	**raptorxmlxbrl valxml-withxsd | xsi [options] *InputFile***
Mac	**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

▼ Casing on the command line

RaptorXMLXBRL *on Windows*
raptorxmlxbrl *on Unix (Linux, Mac)*

* Note that lowercase (raptorxmlxbrl) works on all platforms (Windows, Linux, and Mac), while upper-lower (RaptorXMLXBRL) works only on Windows and Mac.

Options
The command's options are listed below, organized into groups. Values 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.

▼ Validation and processing

▼ **assessment-mode**

--assessment-mode = 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.

listfile

--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.

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

parallel-assessment [pa]

--pa | --parallel-assessment = true|false

If set to true, schema validity assessment is carried out in parallel. This means that if there are more than 128 elements at any level, these elements are processed in parallel using multiple threads. Very large XML files can therefore be processed faster if this option is enabled. Parallel assessment takes place on one hierarchical level at a time, but can occur at multiple levels within a single infoset. Note that parallel assessment does not work in streaming mode. For this reason, the --streaming option is ignored if --parallel-assessment is set to true. Also, memory usage is higher when the --parallel-assessment option is used. The default setting is false. Short form for the option is --pa.

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

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.

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

schema-imports

--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 (XBRL option and XML/XSD 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.

schemalocation-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 (XBRL option and XML/XSD 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-mapping

--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`.

script

--script = *FILE*

Executes the Python script in the submitted file after validation has been completed.

script-param

--script-param = *KEY:VALUE*

Additional user-specified parameters that can be accessed during the execution of Python scripts.

streaming

--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 that the `--streaming` option is ignored if `--parallel-assessment` is set to `true`.

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

xinclude

--xinclude = `true|false`

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

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

xml-mode

--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`.

xsd

--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.

xsd-version

--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 `vc:minVersion` attribute of the document's `<xs:schema>` element. If the value of the `@vc:minVersion` attribute is `1.1`, the schema is detected as being version `1.1`. For any other value, or if the `@vc:minVersion` attribute is absent, the schema is detected as being version `1.0`.

Catalogs and global resources

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 (`<installation-folder>\Altova\RaptorXMLXBRLServer2015\etc\RootCatalog.xml`). See the section, XML Catalogs, for information about working with catalogs.

user-catalog

--user-catalog = *FILE*

Specifies the absolute path to an XML catalog to be used in addition to the root catalog. See the section, XML Catalogs, for information about working with catalogs.

enable-globalresources

--enable-globalresources = `true|false`

Enables global resources. Default value is `false`.

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

globalresourceconfig [gc]

--gc | --globalresourceconfig = *VALUE*

Specifies the active configuration of the global resource (and enables global resources).

globalresourcefile [gr]

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

Messages, errors, help, timeout, version

error-format

--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.

error-limit

--error-limit = *N*
Specifies the error limit. Default value is `100`. Values of `1` to `999` are allowed. Useful for limiting processor use during validation. When the error limit is reached, validation stops.

help

--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`.)

log-output

--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.

network-timeout

--network-timeout = *VALUE*
Specifies the timeout in seconds for remote I/O operations. Default is: `40`.

verbose

--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.

verbose-output

--verbose-output = *FILE*
Writes verbose output to *FILE*.

version

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

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.

Windows	**RaptorXMLXBRL valdtd | dtd [options] *InputFile***
Linux	**raptorxmlxbrl valdtd | dtd [options] *InputFile***
Mac	**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

▼ Casing on the command line

RaptorXMLXBRL *on Windows*
raptorxmlxbrl *on Unix (Linux, Mac)*

* Note that lowercase (raptorxmlxbrl) works on all platforms (Windows, Linux, and Mac), while upper-lower (RaptorXMLXBRL) works only on Windows and Mac.

Options
The command's options are listed below, organized into groups. Values 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.

▼ Validation and processing

▼ **listfile**

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

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

- Catalogs and global resources
  - **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 (`<installation-folder>\Altova\RaptorXMLXBRLServer2015\etc\RootCatalog.xml`). See the section, XML Catalogs, for information about working with catalogs.

  - **user-catalog**

    --user-catalog = *FILE*
    Specifies the absolute path to an XML catalog to be used in addition to the root catalog. See the section, XML Catalogs, for information about working with catalogs.

  - **enable-globalresources**

    --enable-globalresources = `true|false`
    Enables global resources. Default value is `false`.
    Note: Boolean option values are set to `true` if the option is specified without a value.

  - **globalresourceconfig [gc]**

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

  - **globalresourcefile [gr]**

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

- Messages, errors, help, timeout, version
  - **error-format**

    --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.

  - **error-limit**

    --error-limit = *N*
    Specifies the error limit. Default value is `100`. Values of `1` to `999` are allowed. Useful for limiting processor use during validation. When the error limit is reached, validation stops.

  - **help**

    --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`.)

log-output

--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.

network-timeout

--network-timeout = *VALUE*
Specifies the timeout in seconds for remote I/O operations. Default is: `40`.

verbose

--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.

verbose-output

--verbose-output = *FILE*
Writes verbose output to *FILE*.

version

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

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.

Windows **RaptorXMLXBRL valxsd | xsd [options]** ***InputFile***
Linux **raptorxmlxbrl valxsd | xsd [options]** ***InputFile***
Mac **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

▼ Casing on the command line

RaptorXMLXBRL *on Windows*
raptorxmlxbrl *on Unix (Linux, Mac)*

* Note that lowercase (raptorxmlxbrl) works on all platforms (Windows, Linux, and Mac), while upper-lower (RaptorXMLXBRL) works only on Windows and Mac.

Options
The command's options are listed below, organized into groups. Values 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.

▼ Validation and processing

▼ **listfile**

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

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`.

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

schema-imports

--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 (XBRL option and XML/XSD 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.

schemalocation-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 (XBRL option and XML/XSD 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-mapping

--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`.

script

--script = *FILE*

Executes the Python script in the submitted file after validation has been completed.

script-param

--script-param = *KEY:VALUE*

Additional user-specified parameters that can be accessed during the execution of Python scripts.

xinclude

--xinclude = `true|false`

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

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

xml-mode

--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`.

xsd-version

--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 `vc:minVersion` attribute of the document's `<xs:schema>` element. If the value of the `@vc:minVersion` attribute is `1.1`, the schema is detected as being version `1.1`. For any other value, or if the `@vc:minVersion` attribute is absent, the schema is detected as being version `1.0`.

Catalogs and global resources

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 (`<installation-folder>\Altova\RaptorXMLXBRLServer2015\etc\RootCatalog.xml`). See the section, XML Catalogs, for information about working with catalogs.

user-catalog

--user-catalog = *FILE*

Specifies the absolute path to an XML catalog to be used in addition to the root catalog. See the section, XML Catalogs, for information about working with catalogs.

enable-globalresources

--enable-globalresources = `true|false`
Enables global resources. Default value is `false`.
Note: Boolean option values are set to `true` if the option is specified without a value.

globalresourceconfig [gc]

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

globalresourcefile [gr]

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

Messages, errors, help, timeout, version

error-format

--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.

error-limit

--error-limit = *N*
Specifies the error limit. Default value is `100`. Values of `1` to `999` are allowed. Useful for limiting processor use during validation. When the error limit is reached, validation stops.

help

--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`.)

log-output

--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.

network-timeout

--network-timeout = *VALUE*
Specifies the timeout in seconds for remote I/O operations. Default is: `40`.

verbose

--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.

verbose-output

--verbose-output = *FILE*
Writes verbose output to *FILE*.

version

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

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.

```
Windows  RaptorXMLXBRL valany [options] InputFile
Linux    raptorxmlxbrl valany [options] InputFile
Mac      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

▼ Casing on the command line

RaptorXMLXBRL *on Windows*
raptorxmlxbrl *on Unix (Linux, Mac)*

* Note that lowercase (raptorxmlxbrl) works on all platforms (Windows, Linux, and Mac), while upper-lower (RaptorXMLXBRL) works only on Windows and Mac.

Options
The command's options are listed below, organized into groups. Values 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.

▼ Validation and processing

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

▼ **schema-imports**

--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 (XBRL option and XML/XSD 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.

schemalocation-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 (XBRL option and XML/XSD 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-mapping

--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.

Catalogs and global resources

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 (<installation-folder>\Altova\RaptorXMLXBRLServer2015\etc\RootCatalog.xml). See the section, XML Catalogs, for information about working with catalogs.

user-catalog

--user-catalog = *FILE*
Specifies the absolute path to an XML catalog to be used in addition to the root catalog. See the section, XML Catalogs, for information about working with catalogs.

enable-globalresources

--enable-globalresources = `true|false`
Enables global resources. Default value is `false`.
Note: Boolean option values are set to `true` if the option is specified without a value.

globalresourceconfig [gc]

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

globalresourcefile [gr]

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

Messages, errors, help, timeout, version

error-format

--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.

error-limit

--error-limit = *N*
Specifies the error limit. Default value is `100`. Values of `1` to `999` are allowed. Useful for limiting processor use during validation. When the error limit is reached, validation stops.

help

--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`.)

log-output

--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.

network-timeout

--network-timeout = *VALUE*
Specifies the timeout in seconds for remote I/O operations. Default is: `40`.

verbose

--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.

verbose-output

--verbose-output = *FILE*
Writes verbose output to *FILE*.

version

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

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.

Windows **RaptorXMLXBRL wfxml [options] *InputFile***
Linux **raptorxmlxbrl wfxml [options] *InputFile***
Mac **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

▼ Casing on the command line

RaptorXMLXBRL *on Windows*
raptorxmlxbrl *on Unix (Linux, Mac)*

* Note that lowercase (raptorxmlxbrl) works on all platforms (Windows, Linux, and Mac), while upper-lower (RaptorXMLXBRL) works only on Windows and Mac.

Options
The command's options are listed below, organized into groups. Values 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.

▼ Validation and processing

▼ **dtd**

--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.

▼ **listfile**

--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.

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

namespaces

`--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`.

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

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`.

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

Catalogs and global resources

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 (`<installation-folder>\Altova\RaptorXMLXBRLServer2015\etc\RootCatalog.xml`). See the section, XML Catalogs, for information about working with catalogs.

user-catalog

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

Specifies the absolute path to an XML catalog to be used in addition to the root catalog. See the section, XML Catalogs, for information about working with catalogs.

enable-globalresources

`--enable-globalresources =` `true|false`

Enables global resources. Default value is `false`.

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

globalresourceconfig [gc]

`--gc | --globalresourceconfig =` *`VALUE`*

Specifies the active configuration of the global resource (and enables global resources).

globalresourcefile [gr]

`--gr | --globalresourcefile =` *`FILE`*

Specifies the global resource file (and enables global resources).

Messages, errors, help, timeout, version

error-format

--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.

error-limit

--error-limit = *N*
Specifies the error limit. Default value is `100`. Values of `1` to `999` are allowed. Useful for limiting processor use during validation. When the error limit is reached, validation stops.

help

--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`.)

log-output

--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.

network-timeout

--network-timeout = *VALUE*
Specifies the timeout in seconds for remote I/O operations. Default is: `40`.

verbose

--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.

verbose-output

--verbose-output = *FILE*
Writes verbose output to *FILE*.

version

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

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.

Windows **RaptorXMLXBRL wfdtd [options] *InputFile***
Linux **raptorxmlxbrl wfdtd [options] *InputFile***
Mac **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

▼ Casing on the command line

RaptorXMLXBRL *on Windows*
raptorxmlxbrl *on Unix (Linux, Mac)*

* Note that lowercase (raptorxmlxbrl) works on all platforms (Windows, Linux, and Mac), while upper-lower (RaptorXMLXBRL) works only on Windows and Mac.

Options
The command's options are listed below, organized into groups. Values 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.

▼ Validation and processing

▼ **listfile**

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

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

- Catalogs and global resources
  - **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 (`<installation-folder>\Altova\RaptorXMLXBRLServer2015\etc\RootCatalog.xml`). See the section, XML Catalogs, for information about working with catalogs.

  - **user-catalog**

    --user-catalog = *FILE*
    Specifies the absolute path to an XML catalog to be used in addition to the root catalog. See the section, XML Catalogs, for information about working with catalogs.

  - **enable-globalresources**

    --enable-globalresources = `true|false`
    Enables global resources. Default value is `false`.
    Note: Boolean option values are set to `true` if the option is specified without a value.

  - **globalresourceconfig [gc]**

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

  - **globalresourcefile [gr]**

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

- Messages, errors, help, timeout, version
  - **error-format**

    --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.

  - **error-limit**

    --error-limit = *N*
    Specifies the error limit. Default value is `100`. Values of `1` to `999` are allowed. Useful for limiting processor use during validation. When the error limit is reached, validation stops.

  - **help**

    --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`.)

log-output

--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.

network-timeout

--network-timeout = *VALUE*
Specifies the timeout in seconds for remote I/O operations. Default is: `40`.

verbose

--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.

verbose-output

--verbose-output = *FILE*
Writes verbose output to *FILE*.

version

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

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.

Windows **RaptorXMLXBRL wfany [options] *InputFile***
Linux **raptorxmlxbrl wfany [options] *InputFile***
Mac **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

▼ Casing on the command line

RaptorXMLXBRL *on Windows*
raptorxmlxbrl *on Unix (Linux, Mac)*

* Note that lowercase (raptorxmlxbrl) works on all platforms (Windows, Linux, and Mac), while upper-lower (RaptorXMLXBRL) works only on Windows and Mac.

Options
The command's options are listed below, organized into groups. Values 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.

▼ Validation and processing

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

▼ Catalogs and global resources

▼ **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 (`<installation-folder>\Altova\RaptorXMLXBRLServer2015\etc\RootCatalog.xml`). See the section, XML Catalogs, for information about working with catalogs.

user-catalog

--user-catalog = *FILE*
Specifies the absolute path to an XML catalog to be used in addition to the root catalog. See the section, XML Catalogs, for information about working with catalogs.

enable-globalresources

--enable-globalresources = `true|false`
Enables global resources. Default value is `false`.
Note: Boolean option values are set to `true` if the option is specified without a value.

globalresourceconfig [gc]

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

globalresourcefile [gr]

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

Messages, errors, help, timeout, version

error-format

--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.

error-limit

--error-limit = *N*
Specifies the error limit. Default value is `100`. Values of `1` to `999` are allowed. Useful for limiting processor use during validation. When the error limit is reached, validation stops.

help

--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`.)

log-output

--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.

network-timeout

--network-timeout = *VALUE*

Specifies the timeout in seconds for remote I/O operations. Default is: `40`.

verbose

--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.

verbose-output

--verbose-output = *FILE*

Writes verbose output to *FILE*.

version

--version

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

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.

Windows	**RaptorXMLXBRL valxbrl | xbrl [options] *InputFile***
Linux	**raptorxmlxbrl valxbrl | xbrl [options] *InputFile***
Mac	**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

▼ Casing on the command line

RaptorXMLXBRL *on Windows*
raptorxmlxbrl *on Unix (Linux, Mac)*

* Note that lowercase (`raptorxmlxbrl`) works on all platforms (Windows, Linux, and Mac), while upper-lower (`RaptorXMLXBRL`) works only on Windows and Mac.

Options

The command's options are listed below, organized into groups. Values 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 validation and processing

▼ **dimensions**

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

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

listfile

`--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.

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

parallel-assessment [pa]

`--pa | --parallel-assessment =` `true|false`

If set to `true`, schema validity assessment is carried out in parallel. This means that if there are more than 128 elements at any level, these elements are processed in parallel using multiple threads. Very large XML files can therefore be processed faster if this option is enabled. Parallel assessment takes place on one hierarchical level at a time, but can occur at multiple levels within a single infoset. Note that parallel assessment does not work in streaming mode. For this reason, the `--streaming` option is ignored if `--parallel-assessment` is set to `true`. Also, memory usage is higher when the `--parallel-assessment` option is used. The default setting is `false`. Short form for the option is `--pa`.

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

preload-xbrl-schemas

`--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.

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`.

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

schema-imports

`--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 (XBRL option and XML/XSD 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

`--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`.

schemalocation-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 (XBRL option and XML/XSD 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.

script

`--script =` *`FILE`*

Executes the Python script in the submitted file after validation has been completed.

script-param

`--script-param =` *`KEY:VALUE`*

Additional user-specified parameters that can be accessed during the execution of Python scripts.

treat-inconsistencies-as-errors

`--treat-inconsistencies-as-errors =` `true|false`

Causes XBRL validation to fail if the file contains any inconsistency as defined by the XBRL 2.1 specification. Default value is `false`.

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

validate-dts-only

--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.

xinclude

--xinclude = `true|false`
Enables XML Inclusions (XInclude) support. Default value is `false`. When `false`, XInclude's `include` elements are ignored.
Note: Boolean option values are set to `true` if the option is specified without a value.

XBRL formulas and assertions

assertions-output

--assertions-output = *FILE*
Writes the output of the assertion evaluation to the specified *FILE*. If set, automatically specifies `--formula-execution=true`.

assertions-output-format

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

evaluate-referenced-parameters-only

--evaluate-referenced-parameters-only = `true|false`
If `false`, forces evaluation of all parameters even if they are not referenced by any formulas/assertions/tables. Default is: `true`.

formula

--formula = `true|false`
Enables XBRL Formula 1.0 extensions. Default is `true`.
Note: Boolean option values are set to `true` if the option is specified without a value.

formula-assertion-set [[DEPRECATED]]

--formula-assertion-set = *VALUE*
Limits formula execution to the given assertion set only. Add the option multiple times to specify more than one assertion set. Short form is `--as`. The ***VALUE*** is either the value of the `@id` attribute, or a URI with an XPointer fragment that identifies the resource. The special values `##none` and `##all` can also be used.

formula-execution

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

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

formula-output

--formula-output = *FILE*
Writes the output of formula evaluation to the specified *FILE*. If set, automatically specifies `--formula-execution=true`.

formula-parameters

--formula-parameters = *JSON-ARRAY*
Specifies the parameters for XBRL formula evaluation in JSON format directly on the CLI. *See the section, Formula Parameters*. Care must be taken with escaping on the command line.

formula-parameters-file

--formula-parameters-file = *FILE*
Specifies a *FILE* containing the parameters for XBRL formula evaluation. The file can be either an XML file or JSON file. See the section, Formula Parameters.

preload-formula-schemas

--preload-formula-schemas = `true|false`
Preloads schemas of the XBRL Formula 1.0 specification. Default is `false`.
Note: Boolean option values are set to `true` if the option is specified without a value.

process-assertion [a]

--a | --process-assertion = *VALUE*
Limits formula execution to the given assertion only. Add the option multiple times to specify more than one assertion. Short form is `--a`. The ***VALUE*** is either the value of the `@id` attribute, or a URI with an XPointer fragment that identifies the resource. The special values `##none` and `##all` can also be used.

process-assertion-set [as]

--as | --process-assertion-set = *VALUE*
Limits formula execution to the given assertion set only. Add the option multiple times to specify more than one assertion set. Short form is `--as`. The ***VALUE*** is either the value of the `@id` attribute, or a URI with an XPointer fragment that identifies the resource. The special values `##none` and `##all` can also be used.

process-formula [f]

--f | --process-formula = *VALUE*
Limits formula execution to the given formula only. Add the option multiple times to specify more than one formula. Short form is `--f`. The ***VALUE*** is either the value of the `@id` attribute, or a URI with an XPointer fragment that identifies the resource. The special values `##none` and `##all` can also be used.

variableset-execution-timeout

--variableset-execution-timeout = *VALUE*
Applied when executing formulas (**`--formula-execution=true`**). Specifies the maximum time allowed for executing a single variable set (a formula or a value, or an existence or

consistency assertion). The time is specified in minutes and must be a positive number. The default is 30min. If a particular variable set doesn't finish execution before the timeout is reached, then it is aborted. An error message is displayed (and entered in the a verbose log). Note, however, that the timeout check is carried out only after every variable set evaluation—and not during execution of individual XPath expressions. So, if a single XPath expression takes long to execute, the timeout limit might be crossed. Execution of a variable set is aborted only once a complete variable set evaluation has been executed.

- XBRL tables
  - **concept-label-linkrole**

    --concept-label-linkrole = *VALUE*
    Specifies the preferred extended link role to use when rendering concept labels.

  - **concept-label-role**

    --cconcept-label-role = *VALUE*
    Specifies the preferred label role to use when rendering concept labels. Default is: `http://www.xbrl.org/2003/role/label`.

  - **evaluate-referenced-parameters-only**

    --evaluate-referenced-parameters-only = `true|false`
    If `false`, forces evaluation of all parameters even if they are not referenced by any formulas/assertions/tables. Default is: `true`.

  - **generic-label-linkrole**

    --generic-label-linkrole = *VALUE*
    Specifies the preferred extended link role to use when rendering generic labels.

  - **generic-label-role**

    --generic-label-role = *VALUE*
    Specifies the preferred label role to use when rendering generic labels. Default is: http://www.xbrl.org/2003/role/label.

  - **label-lang**

    --label-lang = *VALUE*
    Specifies the preferred label language to use when rendering labels. Default is: `en`.

  - **preload-table-schemas**

    --preload-table-schemas = `true|false`
    Preloads schemas of the XBRL Table 1.0 specification. Default is `false`.
    Note: Boolean option values are set to `true` if the option is specified without a value.

  - **process-table [t]**

    --t | --process-table = *VALUE*
    Limits formula execution to the given table only. Add the option multiple times to specify more than one table. Short form is `--t`. The ***VALUE*** is either the value of the `@id` attribute, or a URI with an XPointer fragment that identifies the resource. The special values `##none`

and ##all can also be used.

table

--table = true|false
Enables the XBRL Table 1.0 extension. Default value is true. If true, automatically specifies --formula=true and --dimensions=true.
Note: Boolean option values are set to true if the option is specified without a value.

table-elimination

--table-elimination = true|false
Enables elimination of empty table rows/columns in HTML output. Default is true.
Note: Boolean option values are set to true if the option is specified without a value.

table-execution

--table-execution = true|false
Enables evaluation of XBRL tables. Default is false. Will be set to true if --table-output is specified. If true, automatically specifies --table=true.
Note: Boolean option values are set to true if the option is specified without a value.

table-linkbase-namespace

--table-linkbase-namespace =
##detect |
http://xbrl.org/PWD/2013-05-17/table |
http://xbrl.org/PWD/2013-08-28/table |
http://xbrl.org/CR/2013-11-13/table |
http://xbrl.org/PR/2013-12-18/table |
http://xbrl.org/2014/table

Enables loading of table linkbases written with a previous draft specification.
Table linkbase validation, resolution, and layout is, however, always performed according to the Table Linkbase 1.0 Recommendation of 18 March 2014. Use ##detect to enable auto-detection.

table-output

--table-output = *FILE*
Writes the table output to the specified *FILE*. If set, automatically specifies --table-execution=true.

table-output-format

--table-output-format = xml|html
Specifies the format of the table output. Default is xml.

Catalogs and global resources

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 (<installation-

folder>\Altova\RaptorXMLXBRLServer2015\etc\RootCatalog.xml). See the section, XML Catalogs, for information about working with catalogs.

- **user-catalog**

  --user-catalog = *FILE*
  Specifies the absolute path to an XML catalog to be used in addition to the root catalog. See the section, XML Catalogs, for information about working with catalogs.

- **enable-globalresources**

  --enable-globalresources = `true|false`
  Enables global resources. Default value is `false`.
  Note: Boolean option values are set to `true` if the option is specified without a value.

- **globalresourceconfig [gc]**

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

- **globalresourcefile [gr]**

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

- Messages, errors, help, timeout, version
  - **error-format**

    --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.

  - **error-limit**

    --error-limit = *N*
    Specifies the error limit. Default value is `100`. Values of `1` to `999` are allowed. Useful for limiting processor use during validation. When the error limit is reached, validation stops.

  - **help**

    --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`.)

  - **log-output**

    --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.

  - **network-timeout**

    --network-timeout = *VALUE*
    Specifies the timeout in seconds for remote I/O operations. Default is: `40`.

verbose

--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.

verbose-output

--verbose-output = *FILE*

Writes verbose output to *FILE*.

version

--version

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

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.

Windows **RaptorXMLXBRL valxbrltaxonomy | dts [options]** ***InputFile***
Linux **raptorxmlxbrl valxbrltaxonomy | dts [options]** ***InputFile***
Mac **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

▼ Casing on the command line

RaptorXMLXBRL *on Windows*
raptorxmlxbrl *on Unix (Linux, Mac)*

* Note that lowercase (`raptorxmlxbrl`) works on all platforms (Windows, Linux, and Mac), while upper-lower (`RaptorXMLXBRL`) works only on Windows and Mac.

Options
The command's options are listed below, organized into groups. Values 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 validation and processing

▼ **dimensions**

--dimensions = `true|false`
Enables XBRL Dimension 1.0 extensions. Default is `true`.
Note: Boolean option values are set to `true` if the option is specified without a value.

▼ **evaluate-referenced-parameters-only**

--evaluate-referenced-parameters-only = `true|false`
If `false`, forces evaluation of all parameters even if they are not referenced by any formulas/assertions/tables. Default is: `true`.

▼ **formula**

--formula = true|false
Enables XBRL Formula 1.0 extensions. Default is `true`.
Note: Boolean option values are set to `true` if the option is specified without a value.

formula-parameters

--formula-parameters = *JSON-ARRAY*
Specifies the parameters for XBRL formula evaluation in JSON format directly on the CLI. *See the section, Formula Parameters*. Care must be taken with escaping on the command line.

formula-parameters-file

--formula-parameters-file = *FILE*
Specifies a *FILE* containing the parameters for XBRL formula evaluation. The file can be either an XML file or JSON file. See the section, Formula Parameters.

listfile

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

preload-formula-schemas

--preload-formula-schemas = true|false
Preloads schemas of the XBRL Formula 1.0 specification. Default is `false`.
Note: Boolean option values are set to `true` if the option is specified without a value.

preload-xbrl-schemas

--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.

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

schema-imports

--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 (XBRL option and XML/XSD 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

`--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`.

schemalocation-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 (XBRL option and XML/XSD 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.

script

`--script =` *`FILE`*

Executes the Python script in the submitted file after validation has been completed.

script-param

`--script-param =` *`KEY:VALUE`*

Additional user-specified parameters that can be accessed during the execution of Python scripts.

- **treat-inconsistencies-as-errors**

  --treat-inconsistencies-as-errors = `true|false`
  Causes XBRL validation to fail if the file contains any inconsistency as defined by the XBRL 2.1 specification. Default value is `false`.
  Note: Boolean option values are set to `true` if the option is specified without a value.

- **xinclude**

  --xinclude = `true|false`
  Enables XML Inclusions (XInclude) support. Default value is `false`. When `false`, XInclude's `include` elements are ignored.
  Note: Boolean option values are set to `true` if the option is specified without a value.

- XBRL tables
  - **concept-label-linkrole**

    --concept-label-linkrole = *VALUE*
    Specifies the preferred extended link role to use when rendering concept labels.

  - **concept-label-role**

    --cconcept-label-role = *VALUE*
    Specifies the preferred label role to use when rendering concept labels. Default is: `http://www.xbrl.org/2003/role/label`.

  - **evaluate-referenced-parameters-only**

    --evaluate-referenced-parameters-only = `true|false`
    If `false`, forces evaluation of all parameters even if they are not referenced by any formulas/assertions/tables. Default is: `true`.

  - **generic-label-linkrole**

    --generic-label-linkrole = *VALUE*
    Specifies the preferred extended link role to use when rendering generic labels.

  - **generic-label-role**

    --generic-label-role = *VALUE*
    Specifies the preferred label role to use when rendering generic labels. Default is: http://www.xbrl.org/2003/role/label.

  - **label-lang**

    --label-lang = *VALUE*
    Specifies the preferred label language to use when rendering labels. Default is: `en`.

  - **preload-table-schemas**

    --preload-table-schemas = `true|false`
    Preloads schemas of the XBRL Table 1.0 specification. Default is `false`.
    Note: Boolean option values are set to `true` if the option is specified without a value.

  - **process-table [t]**

--t | --process-table = *VALUE*
Limits formula execution to the given table only. Add the option multiple times to specify more than one table. Short form is `--t`. The ***VALUE*** is either the value of the `@id` attribute, or a URI with an XPointer fragment that identifies the resource. The special values `##none` and `##all` can also be used.

▼ **table**

--table = `true|false`
Enables the XBRL Table 1.0 extension. Default value is `true`. If `true`, automatically specifies `--formula=true` and `--dimensions=true`.
Note: Boolean option values are set to `true` if the option is specified without a value.

▼ **table-execution**

--table-execution = `true|false`
Enables evaluation of XBRL tables. Default is `false`. Will be set to true if `--table-output` is specified. If `true`, automatically specifies `--table=true`.
Note: Boolean option values are set to `true` if the option is specified without a value.

▼ **table-linkbase-namespace**

--table-linkbase-namespace =
`##detect |`
`http://xbrl.org/PWD/2013-05-17/table |`
`http://xbrl.org/PWD/2013-08-28/table |`
`http://xbrl.org/CR/2013-11-13/table |`
`http://xbrl.org/PR/2013-12-18/table |`
`http://xbrl.org/2014/table`

Enables loading of table linkbases written with a previous draft specification.
Table linkbase validation, resolution, and layout is, however, always performed according to the Table Linkbase 1.0 Recommendation of 18 March 2014. Use `##detect` to enable auto-detection.

▼ **table-output**

--table-output = *FILE*
Writes the table output to the specified *FILE*. If set, automatically specifies `--table-execution=true`.

▼ **table-output-format**

--table-output-format = `xml|html`
Specifies the format of the table output. Default is `xml`.

▼ Catalogs and global resources

▼ **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 (`<installation-folder>\Altova\RaptorXMLXBRLServer2015\etc\RootCatalog.xml`). See the section,

XML Catalogs, for information about working with catalogs.

user-catalog

--user-catalog = *FILE*
Specifies the absolute path to an XML catalog to be used in addition to the root catalog. See the section, XML Catalogs, for information about working with catalogs.

enable-globalresources

--enable-globalresources = `true|false`
Enables global resources. Default value is `false`.
Note: Boolean option values are set to `true` if the option is specified without a value.

globalresourceconfig [gc]

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

globalresourcefile [gr]

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

Messages, errors, help, timeout, version

error-format

--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.

error-limit

--error-limit = *N*
Specifies the error limit. Default value is `100`. Values of `1` to `999` are allowed. Useful for limiting processor use during validation. When the error limit is reached, validation stops.

help

--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`.)

log-output

--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.

network-timeout

--network-timeout = *VALUE*
Specifies the timeout in seconds for remote I/O operations. Default is: `40`.

verbose

--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.

verbose-output

--verbose-output = *FILE*

Writes verbose output to *FILE*.

version

--version

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

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.

Windows **RaptorXMLXBRL valany [options]** ***InputFile***
Linux **raptorxmlxbrl valany [options]** ***InputFile***
Mac **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

▼ Casing on the command line

RaptorXMLXBRL *on Windows*
raptorxmlxbrl *on Unix (Linux, Mac)*

* Note that lowercase (raptorxmlxbrl) works on all platforms (Windows, Linux, and Mac), while upper-lower (RaptorXMLXBRL) works only on Windows and Mac.

Options
The command's options are listed below, organized into groups. Values 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 validation and processing

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

▼ **schema-imports**

--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 (XBRL option and XML/XSD 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.

schemalocation-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 (XBRL option and XML/XSD 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-mapping

`--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`.

Catalogs and global resources

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 (`<installation-folder>\Altova\RaptorXMLXBRLServer2015\etc\RootCatalog.xml`). See the section,

XML Catalogs, for information about working with catalogs.

user-catalog

--user-catalog = *FILE*
Specifies the absolute path to an XML catalog to be used in addition to the root catalog. See the section, XML Catalogs, for information about working with catalogs.

enable-globalresources

--enable-globalresources = `true|false`
Enables global resources. Default value is `false`.
Note: Boolean option values are set to `true` if the option is specified without a value.

globalresourceconfig [gc]

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

globalresourcefile [gr]

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

Messages, errors, help, timeout, version

error-format

--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.

error-limit

--error-limit = *N*
Specifies the error limit. Default value is `100`. Values of `1` to `999` are allowed. Useful for limiting processor use during validation. When the error limit is reached, validation stops.

help

--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`.)

log-output

--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.

network-timeout

--network-timeout = *VALUE*
Specifies the timeout in seconds for remote I/O operations. Default is: `40`.

verbose

--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.

verbose-output

--verbose-output = *FILE*

Writes verbose output to *FILE*.

version

--version

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

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.

Windows **RaptorXMLXBRL xslt [options]** ***XSLT-File***
Linux **raptorxmlxbrl xslt [options]** ***XSLT-File***
Mac **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[1]/@att1 --p=title:'stringwithoutspace' --param=title:"'string with spaces'" --p=amount:456 c:\Test.xslt

▼ Casing on the command line

RaptorXMLXBRL *on Windows*
raptorxmlxbrl *on Unix (Linux, Mac)*

* Note that lowercase (raptorxmlxbrl) works on all platforms (Windows, Linux, and Mac), while upper-lower (RaptorXMLXBRL) works only on Windows and Mac.

Options
The command's options are listed below, organized into groups. Values 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.

▼ XSLT processing

▼ **indent-characters**

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

▼ **input**

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

output

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.

param [p]

--p | --param = *KEY:VALUE*

XQuery

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.

XSLT

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[1]/@att1 --p=title:'stringwithoutspace' --
param=title:"'string with spaces'" --p=amount:456 c:\Test.xslt
```

streaming

--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 that the `--streaming` option is ignored if `--parallel-assessment` is set to `true`.

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

template-entry-point

--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

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

xslt-version

--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.

XML Schema and XML instance

load-xml-with-psvi

--load-xml-with-psvi = true|false
Enables validation of input XML files and generates post-schema-validation information for them. Default is: false.

schema-imports

--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 (XBRL option and XML/XSD 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.

schemalocation-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 (XBRL option and XML/XSD 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-mapping

--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`.

xinclude

--xinclude = `true|false`
Enables XML Inclusions (XInclude) support. Default value is `false`. When `false`, XInclude's `include` elements are ignored.
Note: Boolean option values are set to `true` if the option is specified without a value.

xml-mode

--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`.

xsd-version

--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 `vc:minVersion` attribute of the document's `<xs:schema>` element. If the value of the `@vc:minVersion` attribute is `1.1`, the schema is detected as being version `1.1`. For any other value, or if the `@vc:minVersion` attribute is absent, the schema is detected as being version `1.0`.

Catalogs and global resources

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 (`<installation-folder>\Altova\RaptorXMLXBRLServer2015\etc\RootCatalog.xml`). See the section, XML Catalogs, for information about working with catalogs.

user-catalog

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

See the section, XML Catalogs, for information about working with catalogs.

enable-globalresources

--enable-globalresources = true|false
Enables global resources. Default value is false.
Note: Boolean option values are set to true if the option is specified without a value.

globalresourceconfig [gc]

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

globalresourcefile [gr]

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

Extensions
These options define the handling of special extension functions that are available in a number of Enterprise-level Altova products (such as XMLSpy Enterprise Edition). Their use is described in the user manuals of these products.

chartext-disable

--chartext-disable = true|false
Disables chart extensions. Default value is false.
Note: Boolean option values are set to true if the option is specified without a value.

dotnetext-disable

--dotnetext-disable = true|false
Disables .NET extensions. Default value is false.
Note: Boolean option values are set to true if the option is specified without a value.

javaext-barcode-location

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

javaext-disable

--javaext-disable = true|false
Disables Java extensions. Default value is false.
Note: Boolean option values are set to true if the option is specified without a value.

Messages, errors, help, timeout, version

error-format

--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.

error-limit

--error-limit = *N*
Specifies the error limit. Default value is `100`. Values of `1` to `999` are allowed. Useful for limiting processor use during validation. When the error limit is reached, validation stops.

help

--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`.)

network-timeout

--network-timeout = *VALUE*
Specifies the timeout in seconds for remote I/O operations. Default is: `40`.

verbose

--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.

verbose-output

--verbose-output = *FILE*
Writes verbose output to *FILE*.

version

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

3.4.2 valxslt

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

Windows **RaptorXMLXBRL valxslt [options] *XSLT-File***
Linux **raptorxmlxbrl valxslt [options] *XSLT-File***
Mac **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

▼ Casing on the command line

RaptorXMLXBRL *on Windows*
raptorxmlxbrl *on Unix (Linux, Mac)*

* Note that lowercase (raptorxmlxbrl) works on all platforms (Windows, Linux, and Mac), while upper-lower (RaptorXMLXBRL) works only on Windows and Mac.

Options
The command's options are listed below, organized into groups. Values 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.

▼ XSLT processing

▼ **template-entry-point**

--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**

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

▼ **xslt-version**

--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.

- XML Schema and XML instance
  - **load-xml-with-psvi**

    --load-xml-with-psvi = true|false
    Enables validation of input XML files and generates post-schema-validation information for them. Default is: false.

  - **schema-imports**

    --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 (XBRL option and XML/XSD 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.

  - **schemalocation-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 (XBRL option and XML/XSD 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-mapping**

    --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`.

xinclude

--xinclude = `true|false`
Enables XML Inclusions (XInclude) support. Default value is `false`. When `false`, XInclude's `include` elements are ignored.
Note: Boolean option values are set to `true` if the option is specified without a value.

xml-mode

--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`.

xsd-version

--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 `vc:minVersion` attribute of the document's `<xs:schema>` element. If the value of the `@vc:minVersion` attribute is `1.1`, the schema is detected as being version `1.1`. For any other value, or if the `@vc:minVersion` attribute is absent, the schema is detected as being version `1.0`.

Catalogs and global resources

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 (`<installation-folder>\Altova\RaptorXMLXBRLServer2015\etc\RootCatalog.xml`). See the section, XML Catalogs, for information about working with catalogs.

user-catalog

--user-catalog = *FILE*
Specifies the absolute path to an XML catalog to be used in addition to the root catalog. See the section, XML Catalogs, for information about working with catalogs.

enable-globalresources

--enable-globalresources = `true|false`
Enables global resources. Default value is `false`.
Note: Boolean option values are set to `true` if the option is specified without a value.

globalresourceconfig [gc]

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

globalresourcefile [gr]

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

Extensions
These options define the handling of special extension functions that are available in a number of Enterprise-level Altova products (such as XMLSpy Enterprise Edition). Their use is described in the user manuals of these products.

chartext-disable

--chartext-disable = `true|false`
Disables chart extensions. Default value is `false`.
Note: Boolean option values are set to `true` if the option is specified without a value.

dotnetext-disable

--dotnetext-disable = `true|false`
Disables .NET extensions. Default value is `false`.
Note: Boolean option values are set to `true` if the option is specified without a value.

javaext-barcode-location

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

javaext-disable

--javaext-disable = `true|false`
Disables Java extensions. Default value is `false`.
Note: Boolean option values are set to `true` if the option is specified without a value.

Messages, errors, help, timeout, version

error-format

--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.

error-limit

--error-limit = *N*
Specifies the error limit. Default value is `100`. Values of `1` to `999` are allowed. Useful for limiting processor use during validation. When the error limit is reached, validation stops.

help

--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`.)

network-timeout

--network-timeout = *VALUE*
Specifies the timeout in seconds for remote I/O operations. Default is: `40`.

verbose

--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.

verbose-output

--verbose-output = *FILE*
Writes verbose output to *FILE*.

version

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

3.5 XQuery Commands

The XQuery commands are:

- xquery: for executing XQuery documents, optionally with an input document
- xqueryupdate: for executing an XQuery update, using an XQuery document and, optionally, the input XML document to update
- 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.

Windows **RaptorXMLXBRL xquery [options] *XQuery-File***
Linux **raptorxmlxbrl xquery [options] *XQuery-File***
Mac **raptorxmlxbrl xquery [options] *XQuery-File***

The argument ***XQuery-File*** is the path and name of the XQuery file to be executed. You can use XQuery 1.0 or 3.0. By default XQuery 3.0 is used.

Examples

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

▼ Casing on the command line

RaptorXMLXBRL *on Windows*
raptorxmlxbrl *on Unix (Linux, Mac)*

* Note that lowercase (raptorxmlxbrl) works on all platforms (Windows, Linux, and Mac), while upper-lower (RaptorXMLXBRL) works only on Windows and Mac.

Options
The command's options are listed below, organized into groups. Values 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.

▼ XQuery Processing

▼ **indent-characters**

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

▼ **input**

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

▼ **omit-xml-declaration**

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

output

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

--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

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

output-method

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

param [p]

--p | --param = *KEY:VALUE*

XQuery
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.

XSLT
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[1]/@att1 --p=title:'stringwithoutspace' --
param=title:"'string with spaces'" --p=amount:456 c:\Test.xslt
```

- **xquery-version**

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

- XML Schema and XML instance
  - **load-xml-with-psvi**

    --load-xml-with-psvi = `true|false`
    Enables validation of input XML files and generates post-schema-validation information for them. Default is: `false`.

  - **xinclude**

    --xinclude = `true|false`
    Enables XML Inclusions (XInclude) support. Default value is `false`. When `false`, XInclude's `include` elements are ignored.
    Note: Boolean option values are set to `true` if the option is specified without a value.

  - **xml-mode**

    --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`.

  - **xsd-version**

    --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 `vc:minVersion` attribute of the document's `<xs:schema>` element. If the value of the `@vc:minVersion` attribute is `1.1`, the schema is detected as being version `1.1`. For any other value, or if the `@vc:minVersion` attribute is absent, the schema is detected as being version `1.0`.

- Catalogs and global resources
  - **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 (`<installation-folder>\Altova\RaptorXMLXBRLServer2015\etc\RootCatalog.xml`). See the section, XML Catalogs, for information about working with catalogs.

user-catalog

--user-catalog = *FILE*
Specifies the absolute path to an XML catalog to be used in addition to the root catalog. See the section, XML Catalogs, for information about working with catalogs.

enable-globalresources

--enable-globalresources = `true|false`
Enables global resources. Default value is `false`.
Note: Boolean option values are set to `true` if the option is specified without a value.

globalresourceconfig [gc]

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

globalresourcefile [gr]

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

Messages, errors, help, timeout, version

error-format

--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.

error-limit

--error-limit = *N*
Specifies the error limit. Default value is `100`. Values of `1` to `999` are allowed. Useful for limiting processor use during validation. When the error limit is reached, validation stops.

help

--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`.)

network-timeout

--network-timeout = *VALUE*
Specifies the timeout in seconds for remote I/O operations. Default is: `40`.

verbose

--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.

verbose-output

--verbose-output = *FILE*
Writes verbose output to *FILE*.

version

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

3.5.2 xqueryupdate

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

Windows **RaptorXMLXBRL xqueryupdate [options] *XQuery-File***
Linux **raptorxmlxbrl xqueryupdate [options] *XQuery-File***
Mac **raptorxmlxbrl xqueryupdate [options] *XQuery-File***

The argument ***XQuery-File*** is the path and name of the XQuery file to be executed. You can specify whether XQuery Update 1.0 or 3.0 should be used. By default XQuery Update 3.0 is used.

Examples

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

▼ Casing on the command line

RaptorXMLXBRL *on Windows*
raptorxmlxbrl *on Unix (Linux, Mac)*

* Note that lowercase (raptorxmlxbrl) works on all platforms (Windows, Linux, and Mac), while upper-lower (RaptorXMLXBRL) works only on Windows and Mac.

Options
The command's options are listed below, organized into groups. Values 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.

▼ XQuery Update Processing

▼ **indent-characters**

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

▼ **input**

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

omit-xml-declaration

--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`.

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

output

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

--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

--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`.

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

output-method

--output-method = `xml|html|xhtml|text`

Specifies the output format. Default value is `xml`.

param [p]

--p | --param = *KEY:VALUE*

XQuery

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.

XSLT

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[1]/@att1 --p=title:'stringwithoutspace' --
param=title:"'string with spaces'" --p=amount:456 c:\Test.xslt
```

xquery-update-version

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

keep-formatting

--keep-formatting = true|false
Keeps the formatting of the target document to the maximum extent that this is possible. Default is: true.

updated-xml

--updated-xml = discard|writeback|asmainresult
Specifies how the updated XML file should be handled. The updates can be either:

- discarded and not written to file (discard)
- written back to the input XML file that is specified with the --input option (writeback)
- saved either to standard output or to the location specified in the --output option (if this is defined)

Default is: discard.

XML Schema and XML instance

load-xml-with-psvi

--load-xml-with-psvi = true|false
Enables validation of input XML files and generates post-schema-validation information for them. Default is: false.

xinclude

--xinclude = true|false
Enables XML Inclusions (XInclude) support. Default value is false. When false, XInclude's include elements are ignored.
Note: Boolean option values are set to true if the option is specified without a value.

xml-mode

--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.

xsd-version

--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 `vc:minVersion` attribute of the document's `<xs:schema>` element. If the value of the `@vc:minVersion` attribute is `1.1`, the schema is detected as being version `1.1`. For any other value, or if the `@vc:minVersion` attribute is absent, the schema is detected as being version `1.0`.

- Catalogs and global resources
  - **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 (`<installation-folder>\Altova\RaptorXMLXBRLServer2015\etc\RootCatalog.xml`). See the section, XML Catalogs, for information about working with catalogs.

  - **user-catalog**

    --user-catalog = *FILE*
    Specifies the absolute path to an XML catalog to be used in addition to the root catalog. See the section, XML Catalogs, for information about working with catalogs.

  - **enable-globalresources**

    --enable-globalresources = `true|false`
    Enables global resources. Default value is `false`.
    Note: Boolean option values are set to `true` if the option is specified without a value.

  - **globalresourceconfig [gc]**

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

  - **globalresourcefile [gr]**

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

- Extensions
  These options define the handling of special extension functions that are available in a number of Enterprise-level Altova products (such as XMLSpy Enterprise Edition). Their use is described in the user manuals of these products.
  - **chartext-disable**

    --chartext-disable = `true|false`
    Disables chart extensions. Default value is `false`.
    Note: Boolean option values are set to `true` if the option is specified without a value.

  - **dotnetext-disable**

    --dotnetext-disable = `true|false`

Disables .NET extensions. Default value is `false`.
Note: Boolean option values are set to `true` if the option is specified without a value.

javaext-barcode-location

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

javaext-disable

--javaext-disable = `true|false`
Disables Java extensions. Default value is `false`.
Note: Boolean option values are set to `true` if the option is specified without a value.

Messages, errors, help, timeout, version

error-format

--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.

error-limit

--error-limit = *N*
Specifies the error limit. Default value is `100`. Values of `1` to `999` are allowed. Useful for limiting processor use during validation. When the error limit is reached, validation stops.

help

--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`.)

network-timeout

--network-timeout = *VALUE*
Specifies the timeout in seconds for remote I/O operations. Default is: `40`.

verbose

--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.

verbose-output

--verbose-output = *FILE*
Writes verbose output to *FILE*.

version

--version
Displays the version of RaptorXML+XBRL Server. If used with a command, place `--`

`version` before the command.

3.5.3 valxquery

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

Windows **RaptorXMLXBRL valxquery [options] *XQuery-File***
Linux **raptorxmlxbrl valxquery [options] *XQuery-File***
Mac **raptorxmlxbrl valxquery [options] *XQuery-File***

The ***XQuery-File*** argument 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

▼ Casing on the command line

RaptorXMLXBRL *on Windows*
raptorxmlxbrl *on Unix (Linux, Mac)*

* Note that lowercase (raptorxmlxbrl) works on all platforms (Windows, Linux, and Mac), while upper-lower (RaptorXMLXBRL) works only on Windows and Mac.

Options
The command's options are listed below, organized into groups. Values 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.

▼ XQuery processing

▼ **omit-xml-declaration**

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

▼ **xquery-version**

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

▼ XML Schema and XML instance

▼ **load-xml-with-psvi**

--load-xml-with-psvi = true|false

Enables validation of input XML files and generates post-schema-validation information for them. Default is: `false`.

xinclude

`--xinclude =` `true|false`
Enables XML Inclusions (XInclude) support. Default value is `false`. When `false`, XInclude's `include` elements are ignored.
Note: Boolean option values are set to `true` if the option is specified without a value.

xml-mode

`--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`.

xsd-version

`--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 `vc:minVersion` attribute of the document's `<xs:schema>` element. If the value of the `@vc:minVersion` attribute is `1.1`, the schema is detected as being version `1.1`. For any other value, or if the `@vc:minVersion` attribute is absent, the schema is detected as being version `1.0`.

Catalogs and global resources

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 (`<installation-folder>\Altova\RaptorXMLXBRLServer2015\etc\RootCatalog.xml`). See the section, XML Catalogs, for information about working with catalogs.

user-catalog

`--user-catalog =` *FILE*
Specifies the absolute path to an XML catalog to be used in addition to the root catalog. See the section, XML Catalogs, for information about working with catalogs.

enable-globalresources

`--enable-globalresources =` `true|false`
Enables global resources. Default value is `false`.
Note: Boolean option values are set to `true` if the option is specified without a value.

globalresourceconfig [gc]

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

▼ **globalresourcefile [gr]**

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

▼ Extensions
These options define the handling of special extension functions that are available in a number of Enterprise-level Altova products (such as XMLSpy Enterprise Edition). Their use is described in the user manuals of these products.

▼ **chartext-disable**

--chartext-disable = `true|false`
Disables chart extensions. Default value is `false`.
Note: Boolean option values are set to `true` if the option is specified without a value.

▼ **dotnetext-disable**

--dotnetext-disable = `true|false`
Disables .NET extensions. Default value is `false`.
Note: Boolean option values are set to `true` if the option is specified without a value.

▼ **javaext-barcode-location**

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

▼ **javaext-disable**

--javaext-disable = `true|false`
Disables Java extensions. Default value is `false`.
Note: Boolean option values are set to `true` if the option is specified without a value.

▼ Messages, errors, help, timeout, version

▼ **error-format**

--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.

▼ **error-limit**

--error-limit = *N*
Specifies the error limit. Default value is `100`. Values of `1` to `999` are allowed. Useful for limiting processor use during validation. When the error limit is reached, validation stops.

▼ **help**

--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`.)

▼ **network-timeout**

--network-timeout = *VALUE*

Specifies the timeout in seconds for remote I/O operations. Default is: `40`.

verbose

--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.

verbose-output

--verbose-output = *FILE*

Writes verbose output to *FILE*.

version

--version

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

3.5.4 valxqueryupdate

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

Windows **RaptorXMLXBRL valxqueryupdate [options] *XQuery-File***
Linux **raptorxmlxbrl valxqueryupdate [options] *XQuery-File***
Mac **raptorxmlxbrl valxqueryupdate [options] *XQuery-File***

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

Examples

- **raptorxmlxbrl** valxqueryupdae c:\Test.xqu
- **raptorxmlxbrl** valxqueryupdate --xquery-version=1 c:\Test.xqu

▼ Casing on the command line

RaptorXMLXBRL *on Windows*
raptorxmlxbrl *on Unix (Linux, Mac)*

* Note that lowercase (raptorxmlxbrl) works on all platforms (Windows, Linux, and Mac), while upper-lower (RaptorXMLXBRL) works only on Windows and Mac.

Options
The command's options are listed below, organized into groups. Values 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.

▼ XQuery processing

▼ **omit-xml-declaration**

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

▼ **xquery-update-version**

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

▼ XML Schema and XML instance

▼ **load-xml-with-psvi**

--load-xml-with-psvi = true|false

Enables validation of input XML files and generates post-schema-validation information for them. Default is: `false`.

xinclude

`--xinclude =` `true|false`
Enables XML Inclusions (XInclude) support. Default value is `false`. When `false`, XInclude's `include` elements are ignored.
Note: Boolean option values are set to `true` if the option is specified without a value.

xml-mode

`--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`.

xsd-version

`--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 `vc:minVersion` attribute of the document's `<xs:schema>` element. If the value of the `@vc:minVersion` attribute is `1.1`, the schema is detected as being version `1.1`. For any other value, or if the `@vc:minVersion` attribute is absent, the schema is detected as being version `1.0`.

Catalogs and global resources

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 (`<installation-folder>\Altova\RaptorXMLXBRLServer2015\etc\RootCatalog.xml`). See the section, XML Catalogs, for information about working with catalogs.

user-catalog

`--user-catalog =` *FILE*
Specifies the absolute path to an XML catalog to be used in addition to the root catalog. See the section, XML Catalogs, for information about working with catalogs.

enable-globalresources

`--enable-globalresources =` `true|false`
Enables global resources. Default value is `false`.
Note: Boolean option values are set to `true` if the option is specified without a value.

globalresourceconfig [gc]

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

globalresourcefile [gr]

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

Extensions

These options define the handling of special extension functions that are available in a number of Enterprise-level Altova products (such as XMLSpy Enterprise Edition). Their use is described in the user manuals of these products.

chartext-disable

--chartext-disable = `true|false`
Disables chart extensions. Default value is `false`.
Note: Boolean option values are set to `true` if the option is specified without a value.

dotnetext-disable

--dotnetext-disable = `true|false`
Disables .NET extensions. Default value is `false`.
Note: Boolean option values are set to `true` if the option is specified without a value.

javaext-barcode-location

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

javaext-disable

--javaext-disable = `true|false`
Disables Java extensions. Default value is `false`.
Note: Boolean option values are set to `true` if the option is specified without a value.

Messages, errors, help, timeout, version

error-format

--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.

error-limit

--error-limit = *N*
Specifies the error limit. Default value is `100`. Values of `1` to `999` are allowed. Useful for limiting processor use during validation. When the error limit is reached, validation stops.

help

--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`.)

network-timeout

--network-timeout = *VALUE*

Specifies the timeout in seconds for remote I/O operations. Default is: `40`.

verbose

--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.

verbose-output

--verbose-output = *FILE*

Writes verbose output to *FILE*.

version

--version

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

3.6 Help and License Commands

This section describes two important features of RaptorXML+XBRL Server:

- Help Command: Describes how to display information about available commands, or about a command's arguments and options
- Licensing: Describes how to 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.

Windows **RaptorXMLXBRL help *Command***
Linux **raptorxmlxbrl help *Command***
Mac **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.

- Casing on the command line

  RaptorXMLXBRL *on Windows*
  raptorxmlxbrl *on Unix (Linux, Mac)*

  * Note that lowercase (`raptorxmlxbrl`) works on all platforms (Windows, Linux, and Mac), while upper-lower (`RaptorXMLXBRL`) works only on Windows and Mac.

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.

- Casing on the command line

  RaptorXMLXBRL *on Windows*
  raptorxmlxbrl *on Unix (Linux, Mac)*

* Note that lowercase (`raptorxmlxbrl`) works on all platforms (Windows, Linux, and Mac), while upper-lower (`RaptorXMLXBRL`) works only on Windows and Mac.

3.6.2 License Commands

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

Windows **`RaptorXMLXBRL licenseserver [options] Server-Or-IP-Address`**
Linux **`raptorxmlxbrl licenseserver [options] Server-Or-IP-Address`**
Mac **`raptorxmlxbrl licenseserver [options] Server-Or-IP-Address`**

On successfully registering RaptorXML+XBRL Server 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.

▼ Casing on the command line

`RaptorXMLXBRL` *on Windows*
`raptorxmlxbrl` *on Unix (Linux, Mac)*

* Note that lowercase (`raptorxmlxbrl`) works on all platforms (Windows, Linux, and Mac), while upper-lower (`RaptorXMLXBRL`) works only on Windows and Mac.

Options
The following options are 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+XBRL Server. 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\RaptorXMLXBRLServer2015\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\RaptorXMLXBRLServer2015\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).

Windows **RaptorXMLXBRL exportresourcestrings *LanguageCode XMLOutputFile***
Linux **raptorxmlxbrl exportresourcestrings *LanguageCode XMLOutputFile***
Mac **raptorxmlxbrl exportresourcestrings *LanguageCode XMLOutputFile***

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
```

▼ Casing on the command line

RaptorXMLXBRL *on Windows*
raptorxmlxbrl *on Unix (Linux, Mac)*

* Note that lowercase (raptorxmlxbrl) works on all platforms (Windows, Linux, and Mac), while upper-lower (RaptorXMLXBRL) works only on Windows and Mac.

3.7.2 setdeflang

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

Windows **RaptorXMLXBRL setdeflang | sdl *LangaugeCode***
Linux **raptorxmlxbrl setdeflang | sdl *LangaugeCode***
Mac **raptorxmlxbrl setdeflang | sdl *LangaugeCode***

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

```
raptorxmlxbrl setdeflang de
```

▼ Casing on the command line

RaptorXMLXBRL *on Windows*
raptorxmlxbrl *on Unix (Linux, Mac)*

* Note that lowercase (`raptorxmlxbrl`) works on all platforms (Windows, Linux, and Mac), while upper-lower (`RaptorXMLXBRL`) works only on Windows and Mac.

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, Global Resources, ZIP Files
- Messages, Errors, Help
- Processing
- XBRL
- XML
- XSD
- XQuery
- XSLT

3.8.1 Catalogs, Global Resources, ZIP Files

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 (`<installation-folder>\Altova\RaptorXMLXBRLServer2015\etc\RootCatalog.xml`). See the section, XML Catalogs, for information about working with catalogs.

user-catalog

--user-catalog = *FILE*
Specifies the absolute path to an XML catalog to be used in addition to the root catalog. See the section, XML Catalogs, for information about working with catalogs.

enable-globalresources

--enable-globalresources = `true|false`
Enables global resources. Default value is `false`.
Note: Boolean option values are set to `true` if the option is specified without a value.

globalresourceconfig [gc]

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

globalresourcefile [gr]

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

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

3.8.2 Messages, Errors, Help, Timeout, Version

error-format

--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.

error-limit

--error-limit = *N*
Specifies the error limit. Default value is `100`. Values of `1` to `999` are allowed. Useful for limiting processor use during validation. When the error limit is reached, validation stops.

help

--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`.)

log-output

--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.

network-timeout

--network-timeout = *VALUE*
Specifies the timeout in seconds for remote I/O operations. Default is: `40`.

verbose

--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.

verbose-output

--verbose-output = *FILE*
Writes verbose output to *FILE*.

version

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

3.8.3 Processing

listfile

--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.

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

parallel-assessment [pa]

--pa | --parallel-assessment = true|false

If set to true, schema validity assessment is carried out in parallel. This means that if there are more than 128 elements at any level, these elements are processed in parallel using multiple threads. Very large XML files can therefore be processed faster if this option is enabled. Parallel assessment takes place on one hierarchical level at a time, but can occur at multiple levels within a single infoset. Note that parallel assessment does not work in streaming mode. For this reason, the --streaming option is ignored if --parallel-assessment is set to true. Also, memory usage is higher when the --parallel-assessment option is used. The default setting is false. Short form for the option is --pa.

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

script

--script = *FILE*

Executes the Python script in the submitted file after validation has been completed.

streaming

--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 that the --streaming option is ignored if --parallel-assessment is set to true.

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

3.8.4 XBRL

XBRL validation and processing options

dimensions

--dimensions = true|false
Enables XBRL Dimension 1.0 extensions. Default is true.
Note: Boolean option values are set to true if the option is specified without a value.

evaluate-referenced-parameters-only

--evaluate-referenced-parameters-only = true|false
If false, forces evaluation of all parameters even if they are not referenced by any formulas/assertions/tables. Default is: true.

listfile

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

parallel-assessment [pa]

--pa | --parallel-assessment = true|false
If set to true, schema validity assessment is carried out in parallel. This means that if there are more than 128 elements at any level, these elements are processed in parallel using multiple threads. Very large XML files can therefore be processed faster if this option is enabled. Parallel assessment takes place on one hierarchical level at a time, but can occur at multiple levels within a single infoset. Note that parallel assessment does not work in streaming mode. For this reason, the --streaming option is ignored if --parallel-assessment is set to true. Also, memory usage is higher when the --parallel-assessment option is used. The default setting is false. Short form for the option is --pa.
Note: Boolean option values are set to true if the option is specified without a value.

preload-xbrl-schemas

--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.

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

schema-imports

--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 (XBRL option and XML/XSD 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

--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.

schemalocation-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 (XBRL option and XML/XSD 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.

script

--script = *FILE*

Executes the Python script in the submitted file after validation has been completed.

script-param

--script-param = *KEY:VALUE*
Additional user-specified parameters that can be accessed during the execution of Python scripts.

treat-inconsistencies-as-errors

--treat-inconsistencies-as-errors = `true|false`
Causes XBRL validation to fail if the file contains any inconsistency as defined by the XBRL 2.1 specification. Default value is `false`.
Note: Boolean option values are set to `true` if the option is specified without a value.

validate-dts-only

--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.

xinclude

--xinclude = `true|false`
Enables XML Inclusions (XInclude) support. Default value is `false`. When `false`, XInclude's `include` elements are ignored.
Note: Boolean option values are set to `true` if the option is specified without a value.

XBRL formulas and assertion options

assertions-output

--assertions-output = *FILE*
Writes the output of the assertion evaluation to the specified *FILE*. If set, automatically specifies `--formula-execution=true`.

assertions-output-format

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

formula

--formula = `true|false`
Enables XBRL Formula 1.0 extensions. Default is `true`.
Note: Boolean option values are set to `true` if the option is specified without a value.

formula-assertion-set [[DEPRECATED]]

--formula-assertion-set = *VALUE*
Limits formula execution to the given assertion set only. Add the option multiple times to specify more than one assertion set. Short form is `--as`. The ***VALUE*** is either the value of the `@id` attribute, or a URI with an XPointer fragment that identifies the resource. The special values `##none` and `##all` can also be used.

formula-execution

--formula-execution = `true|false`
Enables evaluation of XBRL formulas. Default is `true`. If `true`, automatically specifies `--formula=true`.
Note: Boolean option values are set to `true` if the option is specified without a value.

formula-output

--formula-output = *FILE*
Writes the output of formula evaluation to the specified *FILE*. If set, automatically specifies `--formula-execution=true`.

formula-parameters

--formula-parameters = *JSON-ARRAY*
Specifies the parameters for XBRL formula evaluation in JSON format directly on the CLI. *See the section, Formula Parameters*. Care must be taken with escaping on the command line.

formula-parameters-file

--formula-parameters-file = *FILE*
Specifies a *FILE* containing the parameters for XBRL formula evaluation. The file can be either an XML file or JSON file. See the section, Formula Parameters.

preload-formula-schemas

--preload-formula-schemas = `true|false`
Preloads schemas of the XBRL Formula 1.0 specification. Default is `false`.
Note: Boolean option values are set to `true` if the option is specified without a value.

process-assertion [a]

--a | --process-assertion = *VALUE*
Limits formula execution to the given assertion only. Add the option multiple times to specify more than one assertion. Short form is `--a`. The ***VALUE*** is either the value of the `@id` attribute, or a URI with an XPointer fragment that identifies the resource. The special values `##none` and `##all` can also be used.

process-assertion-set [as]

--as | --process-assertion-set = *VALUE*
Limits formula execution to the given assertion set only. Add the option multiple times to specify more than one assertion set. Short form is `--as`. The ***VALUE*** is either the value of the `@id` attribute, or a URI with an XPointer fragment that identifies the resource. The special values `##none` and `##all` can also be used.

process-formula [f]

--f | --process-formula = *VALUE*
Limits formula execution to the given formula only. Add the option multiple times to specify more than one formula. Short form is `--f`. The ***VALUE*** is either the value of the `@id` attribute, or a URI with an XPointer fragment that identifies the resource. The special values `##none` and `##all` can also be used.

variableset-execution-timeout

--variableset-execution-timeout = *VALUE*

Applied when executing formulas (**--formula-execution=true**). Specifies the maximum time allowed for executing a single variable set (a formula or a value, or an existence or consistency assertion). The time is specified in minutes and must be a positive number. The default is 30min. If a particular variable set doesn't finish execution before the timeout is reached, then it is aborted. An error message is displayed (and entered in the a verbose log). Note, however, that the timeout check is carried out only after every variable set evaluation—and not during execution of individual XPath expressions. So, if a single XPath expression takes long to execute, the timeout limit might be crossed. Execution of a variable set is aborted only once a complete variable set evaluation has been executed.

XBRL table options

concept-label-linkrole

--concept-label-linkrole = *VALUE*

Specifies the preferred extended link role to use when rendering concept labels.

concept-label-role

--cconcept-label-role = *VALUE*

Specifies the preferred label role to use when rendering concept labels. Default is: `http://www.xbrl.org/2003/role/label`.

evaluate-referenced-parameters-only

--evaluate-referenced-parameters-only = `true|false`

If `false`, forces evaluation of all parameters even if they are not referenced by any formulas/assertions/tables. Default is: `true`.

generic-label-linkrole

--generic-label-linkrole = *VALUE*

Specifies the preferred extended link role to use when rendering generic labels.

generic-label-role

--generic-label-role = *VALUE*

Specifies the preferred label role to use when rendering generic labels. Default is: http://www.xbrl.org/2003/role/label.

label-lang

--label-lang = *VALUE*

Specifies the preferred label language to use when rendering labels. Default is: `en`.

preload-table-schemas

--preload-table-schemas = `true|false`

Preloads schemas of the XBRL Table 1.0 specification. Default is `false`.

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

process-table [t]

--t | --process-table = *VALUE*
Limits formula execution to the given table only. Add the option multiple times to specify more than one table. Short form is `--t`. The ***VALUE*** is either the value of the `@id` attribute, or a URI with an XPointer fragment that identifies the resource. The special values `##none` and `##all` can also be used.

table

--table = `true|false`
Enables the XBRL Table 1.0 extension. Default value is `true`. If `true`, automatically specifies `--formula=true` and `--dimensions=true`.
Note: Boolean option values are set to `true` if the option is specified without a value.

table-elimination

--table-elimination = `true|false`
Enables elimination of empty table rows/columns in HTML output. Default is `true`.
Note: Boolean option values are set to `true` if the option is specified without a value.

table-execution

--table-execution = `true|false`
Enables evaluation of XBRL tables. Default is `false`. Will be set to true if `--table-output` is specified. If `true`, automatically specifies `--table=true`.
Note: Boolean option values are set to `true` if the option is specified without a value.

table-linkbase-namespace

--table-linkbase-namespace =
`##detect` |
`http://xbrl.org/PWD/2013-05-17/table` |
`http://xbrl.org/PWD/2013-08-28/table` |
`http://xbrl.org/CR/2013-11-13/table` |
`http://xbrl.org/PR/2013-12-18/table` |
`http://xbrl.org/2014/table`

Enables loading of table linkbases written with a previous draft specification.
Table linkbase validation, resolution, and layout is, however, always performed according to the Table Linkbase 1.0 Recommendation of 18 March 2014. Use `##detect` to enable auto-detection.

table-output

--table-output = *FILE*
Writes the table output to the specified *FILE*. If set, automatically specifies `--table-execution=true`.

table-output-format

--table-output-format = `xml|html`
Specifies the format of the table output. Default is `xml`.

3.8.5 XML

assessment-mode

--assessment-mode = 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.

dtd

--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.

load-xml-with-psvi

--load-xml-with-psvi = true|false
Enables validation of input XML files and generates post-schema-validation information for them. Default is: `false`.

namespaces

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

xinclude

--xinclude = true|false
Enables XML Inclusions (XInclude) support. Default value is `false`. When `false`, XInclude's `include` elements are ignored.
Note: Boolean option values are set to `true` if the option is specified without a value.

xml-mode

--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`.

xsd

--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.

3.8.6 XSD

assessment-mode

--assessment-mode = 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.

namespaces

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

schema-imports

--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 (XBRL option and XML/XSD 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.

schemalocation-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 (XBRL option and XML/XSD 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-mapping

`--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

`--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 `vc:minVersion` attribute of the document's `<xs:schema>` element. If the value of the `@vc:minVersion` attribute is `1.1`, the schema is detected as being version `1.1`. For any other value, or if the `@vc:minVersion` attribute is absent, the schema is detected as being version `1.0`.

3.8.7 XQuery

indent-characters

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

input

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

keep-formatting

--keep-formatting = `true|false`
Keeps the formatting of the target document to the maximum extent that this is possible. Default is: `true`.

omit-xml-declaration

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

output

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

--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

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

output-method

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

param [p]

--p | --param = *KEY:VALUE*

XQuery
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.

XSLT

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[1]/@att1 --p=title:'stringwithoutspace' --
param=title:"'string with spaces'" --p=amount:456 c:\Test.xslt
```

updated-xml

--updated-xml = `discard|writeback|asmainresult`

Specifies how the updated XML file should be handled. The updates can be either:

- discarded and not written to file (`discard`)
- written back to the input XML file that is specified with the `--input` option (`writeback`)
- saved either to standard output or to the location specified in the `--output` option (if this is defined)

Default is: `discard`.

xquery-update-version

--xquery-update-version = `1|3`

Specifies whether the XQuery processor should use XQuery Update Facility 1.0 or XQuery Update Facility 3.0. Default value is `3`.

xquery-version

--xquery-version = `1|3`

Specifies whether the XQuery processor should use XQuery 1.0 or XQuery 3.0. Default value is 3.

3.8.8 XSLT

chartext-disable

--chartext-disable = `true|false`

Disables chart extensions. Default value is `false`.

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

dotnetext-disable

--dotnetext-disable = `true|false`

Disables .NET extensions. Default value is `false`.

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

indent-characters

--indent-characters = *VALUE*

Specifies the character string to be used as indentation.

input

--input = *FILE*

The URL of the XML file to be transformed.

javaext-barcode-location

--javaext-barcode-location = *FILE*

Specifies the location of the barcode extension file.

javaext-disable

--javaext-disable = `true|false`

Disables Java extensions. Default value is `false`.

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

output

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.

param [p]

--p | --param = *KEY:VALUE*

XQuery

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.

- ***XSLT***
  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[1]/@att1 --p=title:'stringwithoutspace' --
param=title:"'string with spaces'" --p=amount:456 c:\Test.xslt
```

streaming

--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 that the `--streaming` option is ignored if `--parallel-assessment` is set to `true`.
Note: Boolean option values are set to `true` if the option is specified without a value.

template-entry-point

--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

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

xslt-version

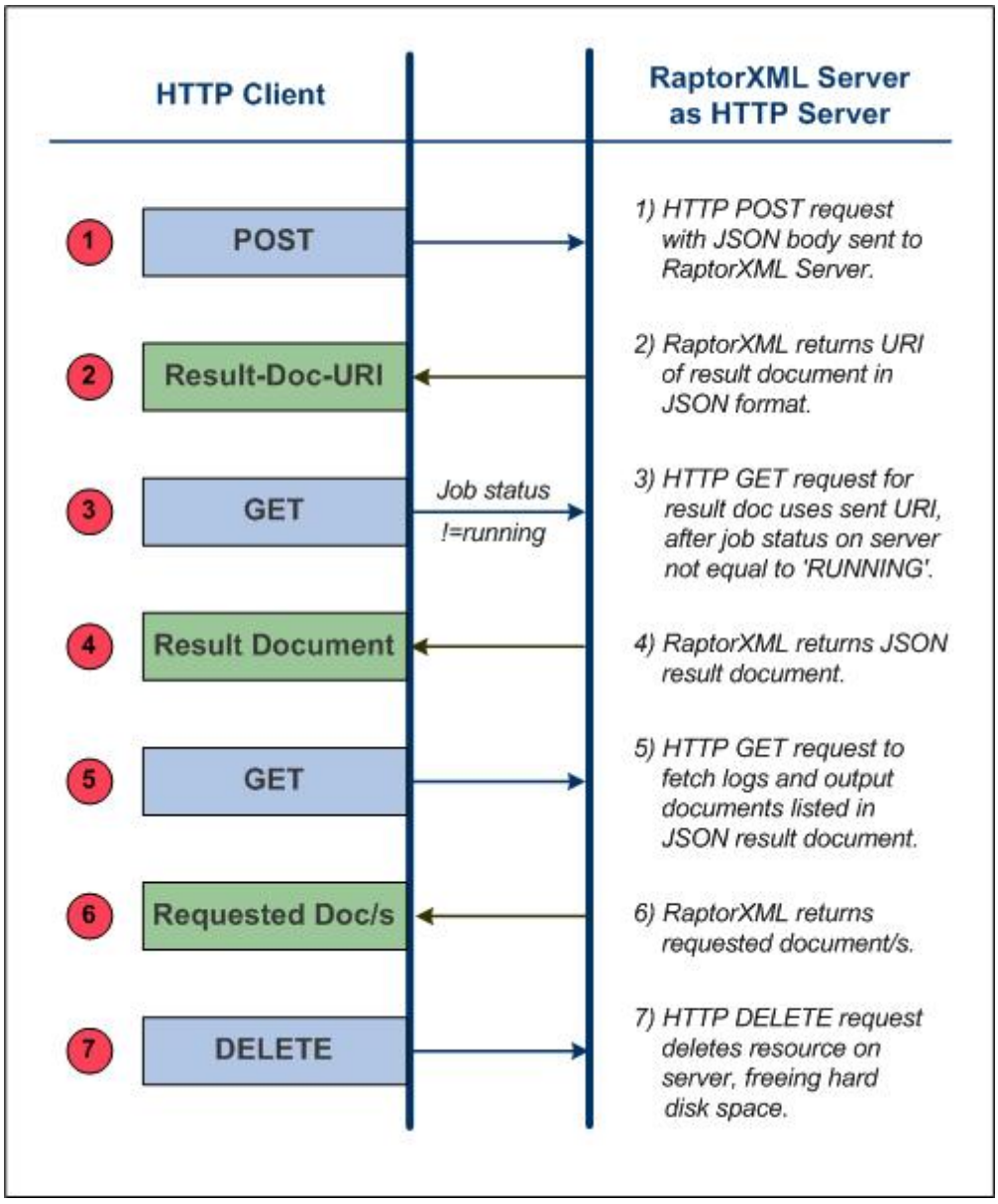
--xslt-version = `1|2|3`
Specifies whether the XSLT processor should use XSLT 1.0, XSLT 2.0, or XSLT 3.0. Default value is `3`.

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 RaptorXML+XBRL 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\RaptorXMLXBRLServer2015\bin\RaptorXMLXBRL.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]   sudo initctl start raptorxmlxbrlserver
```

```
[CentOS]   sudo initctl start raptorxmlxbrlserver
[RedHat]   sudo initctl start raptorxmlxbrlserver
```

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

```
[Debian]   sudo /etc/init.d/raptorxmlxbrlserver stop
[Ubuntu]   sudo initctl stop raptorxmlxbrlserver
[CentOS]   sudo initctl stop raptorxmlxbrlserver
[RedHat]   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.RaptorXMLXBRLServer2015.plist
```

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

```
sudo launchctl unload /Library/LaunchDaemons/
com.altova.RaptorXMLXBRLServer2015.plist
```

4.1.2 Testing the Connection

This section:

- GET request to test the connection
- Server response and JSON data structure listing

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\RaptorXMLXBRLServer2015\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\RaptorXMLXBRLServer2015\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`:

```
RaptorXMLXBRL.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\RaptorXMLXBRLServer2015\Output
  \</server.output-root-dir>
  <server.script-root-dir>C:\Program Files (x86)\Altova\RaptorXMLXBRLServer2015
  \etc\scripts\</server.script-root-dir>
  <!--<server.catalog-file>catalog.xml</server.catalog-file>-->
  <server.log-file>C:\ProgramData\Altova\RaptorXMLXBRLServer2015\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\RaptorXMLXBRLServer2015\Log
  \access.log</http.access-file>
  <http.error-file>C:\ProgramData\Altova\RaptorXMLXBRLServer2015\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\RaptorXMLXBRLServer2015\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. In this case, ensure that firewall settings are suitably configured. Inbound firewall exceptions for Altova products must be registered as follows: Altova LicenseServer: port `8088`; Altova RaptorXML+XBRL Server: port `8087`; Altova FlowForce Server: port `8082`.

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).

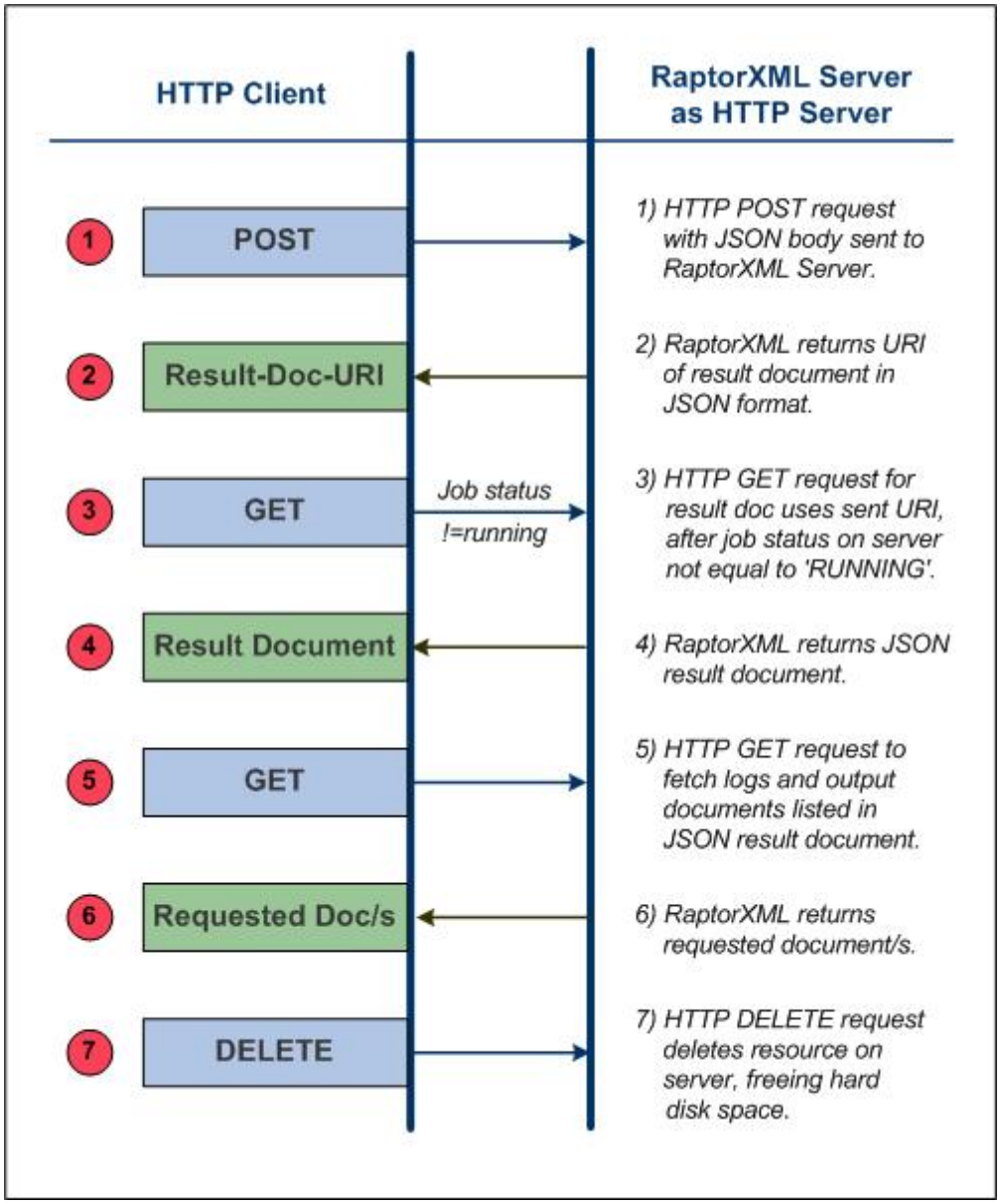
4.2 Client Requests

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 processing 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
- JSON syntax for POST requests
- Uploading files with the POST request

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.

Here is an example with options. Notice that some options (like `input` or `xslt-version`) take a straight option value, while others (like `param`) take a key-value pair as their value, and therefore require a different syntax.

```
{
   "command": "xslt",
   "args": [
           "file:///C:/Work/Test.xslt"
           ],
   "options": {
              "input": "file:///C:/Work/Test.xml",
              "xslt-version": 1,
              "param": {
                       "key": "myTestParam",
                       "value": "SomeParamValue"
                       },
              "output": "file:///C:/temp/out2.xml"
              }
}
```


The example below shows a third type of option: that of an array of values (as for the `xsd` option below). In this case, the syntax to be used is that of a JSON Array.

```
{
   "command": "xsi",
   "args": [
           "file:///C:/Work/Test.xml"
           ],
   "options": {
              "xsd" : ["file:///C:/Work/File1.xsd", "file:///C:/Work/File2.xsd"]
              }
}
```

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: attachment; filename="First.xml"; name="args"
Content-Type: application/octet-stream

  <?xml version="1.0" encoding="UTF-8"?>                                 4
  <test xsi:noNamespaceSchemaLocation="Demo.xsd" xmlns:xsi="http://
  www.w3.org/2001/XMLSchema-instance">42</test>

---PartBoundary                                                          5
Content-Disposition: attachment; filename="Second.xml"; name="args"
Content-Type: application/octet-stream

  <?xml version="1.0" encoding="UTF-8"?>                                 6
  <test xsi:noNamespaceSchemaLocation="Demo.xsd" xmlns:xsi="http://
  www.w3.org/2001/XMLSchema-instance">35</test>

---PartBoundary                                                          7
Content-Disposition: attachment; filename="Demo.xsd";
name="additional_files"
Content-Type: application/octet-stream

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

---PartBoundary--                                                        9
```

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

4 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.

5 The second member of the `args` array is an attachment called `Second.xml`.

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

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

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

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

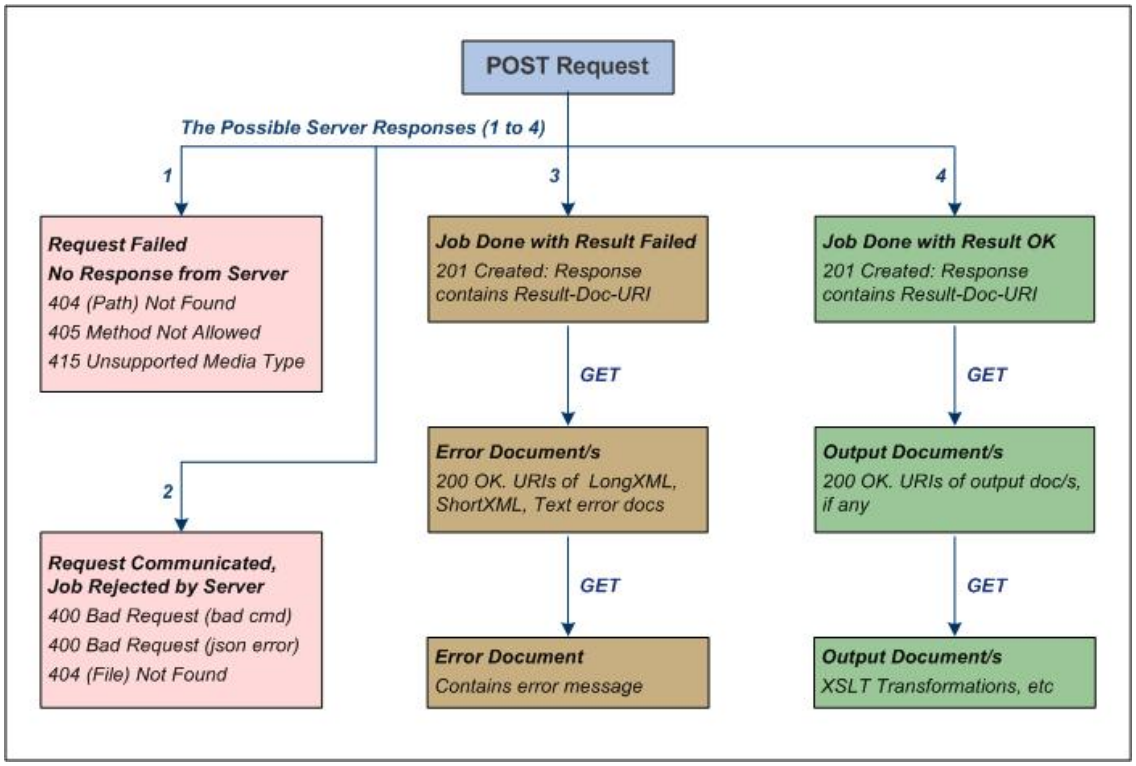
4.2.2 Server Response to POST Request

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 in a command via HTTP to RaptorXML+XBRL 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** at the minimum.

Making Python scripts safe
When a Python script is specified in a command via HTTP to RaptorXML+XBRL 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 RaptorXML+XBRL 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.name
    +'</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.

Job.script_params
Returns a dict (a built-in Python data structure) with the user-specified script parameters (using the `--script-param` option).

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.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.

5.7.2 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.

5.7.3 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.

5.7.4 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.

5.7.5 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.

5.7.6 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.

5.7.7 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.

5.7.8 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.

5.7.9 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.

5.7.10 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.

5.7.11 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.

5.7.12 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`.

5.8 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. For table linkbases, the current RaptorXML+XBRL Server implementation follows the Table Linkbase 1.0 Recommendation of 18 March 2014, and uses the namespace `http://xbrl.org/2014/table`.

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

class xbrl.BreakdownResource
The `BreakdownResource` class represents a breakdown resource in the table linkbase.

class xbrl.Concept
The `Concept` class represents an XBRL concept in the DTS.

class xbrl.ConceptAspectValue
The `ConceptAspectValue` class represents a value for the concept aspect.

class xbrl.ConstraintSet
The `ConstraintSet` class represents a set of constraints for the aspects in the dimensional aspect model.

class xbrl.Context
The `Context` class represents an XBRL context in the instance document.

class xbrl.DefinitionNodeResource
The `DefinitionNodeResource` class represents a `definitionNode` resource in the table linkbase.

class xbrl.DTS
The `DTS` class represents an XBRL Discovery Taxonomy Set (DTS).

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.EntityIdentifierAspectValue
The `EntityIdentifierAspectValue` class represents a value for the entity identifier aspect.

class xbrl.Error
The `Error` class is the base class for any XBRL-related errors.

class xbrl.ExplicitDimensionAspectValue
The `ExplicitDimensionAspectValue` class represents a value for the dimension aspect.

class xbrl.Fact

The `Fact` class represents a fact element in an XBRL instance document.

class xbrl.FootnoteResource
The `FootnoteResource` class represents a footnote resource.

class xbrl.FactSet
The `FactSet` class represents a set of XBRL instance facts.

class xbrl.Fraction
The `Fraction` class represents the fraction value of an XBRL fact item of type.

class xbrl.Instance
The `Instance` class represents an XBRL instance document.

class xbrl.LabelResource
The `LabelResource` class represents a label resource.

class xbrl.LayoutCell
The `LayoutCell` class represents a cell in the generated table.

class xbrl.LayoutDataCell
The `LayoutDataCell` class represents a data cell in the generated table.

class xbrl.LayoutHeaderCell
The `LayoutHeaderCell` class represents a header cell in the generated table.

class xbrl.LayoutRow
The `LayoutRow` class represents a row of the generated table.

class xbrl.LayoutTable
The `LayoutTable` class represents the table after applying the resolution and layout process.

class xbrl.LayoutTableSet
The `LayoutTableSet` class represents the table set after applying the resolution and layout process.

class xbrl.Period
The `Period` class represents the period part of an XBRL context.

class xbrl.PeriodAspectValue
The `PeriodAspectValue` class represents a value for the period aspect.

class xbrl.ReferencePart
The `ReferencePart` class represents a reference part.

class xbrl.ReferenceResource
The `ReferenceResource` class represents a reference resource.

class xbrl.Resource
The `Resource` class represents an XLink resource element within an extended link.

class xbrl.ScenarioAspectValue
The `ScenarioAspectValue` class represents a value for the non-XDT scenario aspect.

class xbrl.SegmentAspectValue
The `SegmentAspectValue` class represents a value for the non-XDT segment aspect.

class xbrl.TableError
The `TableError` class represents an error that is raised during the table resolution or layout process.

class xbrl.TableResource
The `TableResource` class represents a table resource in the table linkbase.

class xbrl.TypedDimensionAspectValue
The `TypedDimensionAspectValue` class represents a value for the dimension aspect.

class xbrl.Unit
The `Unit` class represents an XBRL unit.

class xbrl.UnitAspectValue
The `UnitAspectValue` class represents a value for the unit aspect.

5.9.1 xbrl.BreakdownResource

class **xbrl.BreakdownResource(xbrl.Resource)**
The BreakdownResource class represents a breakdown resource in the table linkbase (http://www.xbrl.org/specification/table-linkbase/pr-2013-12-18/table-linkbase-pr-2013-12-18.html#sec-definition-breakdowns).

5.9.2 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.id
Returns the **id** attribute value as a string, or `"None"` if the XML element doesn't have an `id`

attribute.

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 generator object of xml.Element objects representing all label resources that are associated with this concept.

Concept.labels
Returns a generator object of xbrl.LabelResource objects for each label that has a concept-label (http://www.xbrl.org/2003/arcrole/concept-label) relationship to this concept.

Concept.reference_elements
Returns a generator object of xml.Element objects representing all reference resources that are associated with this concept.

Concept.references
Returns a generator object of xbrl.ReferenceResource objects for each label that has a concept-reference (http://www.xbrl.org/2003/arcrole/concept-reference) relationship to 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`.

Concept.select_labels(label_role = None, link_role = None, lang = None)
Returns a generator object of `xbrl.LabelResource` objects for each label that has a concept-label relationship (`http://www.xbrl.org/2003/arcrole/concept-label`) to this concept and also matches the given parameters.

Concept.select_references(reference_role = None, link_role = None)
Returns a generator object of `xbrl.ReferenceResource` objects for each reference that has a concept-reference relationship (`http://www.xbrl.org/2003/arcrole/concept-reference`) to this concept and also matches the given parameters.

5.9.3 xbrl.ConceptAspectValue

class **xbrl.ConceptAspectValue**
The ConceptAspectValue class represents a value for the concept aspect. The constructor takes the following arguments:

```
ConceptAspectValue(DTS dts, Concept concept = None)
```

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

ConceptAspectValue.qname
Returns an xml.QName object representing the concept's QName.

ConceptAspectValue.concept
Returns an xbrl.Concept object.

5.9.4 xbrl.ConstraintSet

class **xbrl.ConstraintSet(Instance instance)**
The ConstraintSet class represents a set of constraints for the aspects in the dimensional aspect model. For each aspect, zero or one aspect can be specified.

The ConstraintSet class provides the following **constructor**:

ConstraintSet.ConstraintSet
Creates a new empty xbrl.ConstraintSet object. It takes an xbrl.DTS or xbrl.Instance object as as a parameter.

The ConstraintSet class provides the following **instance attributes**:

ConstraintSet.allow_additional_dimensions
Read/write of type Boolean value. If set to false, restricts the facts returned by Instance.select_facts() and FactSet.filter() methods to contain only facts whose assigned contexts don't contain any additional dimensions.

ConstraintSet.concept_aspect
Returns an xbrl.ConceptAspectValue object representing the constraint for the concept aspect, or None if no constraint was specified. Is also writable.

ConstraintSet.entity_identifier_aspect
Returns an xbrl.EntityIdentifierAspectValue object representing the constraint for the entity identifier aspect, or None if no constraint was specified. Is also writable.

ConstraintSet.period_aspect
Returns an xbrl.PeriodAspectValue object representing the constraint for the period aspect, or None if no constraint was specified. Is also writable.

ConstraintSet.scenario_aspect
Returns an xbrl.ScenarioAspectValue object representing the constraint for the non-XDT scenario aspect, or None if no constraint was specified. Is also writable.

ConstraintSet.segment_aspect
Returns an xbrl.SegmentAspectValue object representing the constraint for the non-XDT segment aspect, or None if no constraint was specified. Is also writable.

ConstraintSet.unit_aspect
Returns an xbrl.UnitAspectValue object representing the constraint for the unit aspect, or None if no constraint was specified. Is also writable.

ConstraintSet.dimension_aspects
Returns a generator object of xbrl.ExplicitDimensionAspectValue and xbrl.TypedDimensionAspectValue objects representing the constraints for the dimension aspect.

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

ConstraintSet.copy()
Returns a new xbrl.ConstraintSet object containing the same constraints.

ConstraintSet.clear()
Removes any constraints for all aspects.

ConstraintSet.append(aspectValue)
Adds a new constraint. Any existing constraint for this aspect will be overwritten.

ConstraintSet.extend(constraintSet)
Adds all the constraints from xbrl.ConstraintSet constraint-set. Any existing constraints will be overwritten.

ConstraintSet.dimension_aspect(dimension)
Returns an xbrl.ExplicitDimensionAspectValue or xbrl.TypedDimensionAspectValue object for the given dimension aspect, or None if no constraint was specified.

ConstraintSet.set_dimension_aspect(dimensionAspectValue)
Sets a constraint for the dimension aspect. Any existing constraint for this aspect will be overwritten.

5.9.5 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.

5.9.6 xbrl.DefinitionNodeResource

class **xbrl.DefinitionNodeResource(xbrl.Resource)**
The DefinitionNodeResource class represents a definitionNode resource in the table linkbase (http://www.xbrl.org/specification/table-linkbase/pr-2013-12-18/table-linkbase-pr-2013-12-18.html#sec-definition-nodes).

5.9.7 xbrl.DTS

class **xbrl.DTS**
A DTS class represents an XBRL Discoverable Taxonomy Set (DTS).

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

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`.

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.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.linkbases
Returns a list of xml.Document objects which represent all XBRL linkbases in the DTS.

DTS.schema
Returns an xsd.Schema object which represents the XML Schema component of the underlying DTS.

DTS.schemas
Returns a list of xml.Document objects which represent all XBRL taxonomy schemas in the DTS.

DTS.tables
Returns a generator object of xbrl.TableResource objects which represent all the table resources in the table linkbase.

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`.

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`.

DTS.resolve_table(identifier)
Returns a list of xbrl.TableResource objects that match the identifier. The identifier can be either the value of the `id` attribute or an XPointer URL pointing to the table resource.

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.aspect_value
Returns an xbrl.EntityIdentifierAspectValue object representing the value of the entity identifier aspect.

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.EntityIdentifierAspectValue

class **xbrl.EntityIdentifierAspectValue**
The EntityIdentifierAspectValue class represents a value for the entity identifier aspect. The constructor takes the following arguments:

```
EntityIdentifierAspectValue( DTS dts, identifier = None, scheme = None)
```

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

EntityIdentifierAspectValue.identifier
Returns the identifier as a string.

EntityIdentifierAspectValue.scheme
Returns the identifier scheme as a string.

5.9.11 xbrl.Error

class **xbrl.Error(Exception)**
This is the base class for any XBRL-related errors.

5.9.12 xbrl.ExplicitDimensionAspectValue

class **xbrl.ExplicitDimensionAspectValue**
The ExplicitDimensionAspectValue class represents a value for the dimension aspect. The constructor takes the following arguments:

```
ExplicitDimensionAspectValue( DTS dts, dimension = None, domainMember = None )
```

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

ExplicitDimensionAspectValue.is_explicit
Returns `True`.

ExplicitDimensionAspectValue.is_typed
Returns `False`.

ExplicitDimensionAspectValue.dimension
Returns an xbrl.Concept object representing the dimension.

ExplicitDimensionAspectValue.value
Returns an xbrl.Concept object representing the dimension's domain member value. The absent dimension aspect value is denoted by `None`.

5.9.13 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.id
Returns the **id** attribute value as a string, or "None" if the XML element doesn't have an id attribute.

Fact.footnotes
Returns a generator object of xbrl.FootnoteResource objects for each footnote that has a concept-label relationship (http://www.xbrl.org/2003/arcrole/fact-footnote) to this 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 generator object of Fact objects representing all XBRL facts which are direct children of this fact.

Fact.child_items
Returns a generator object of Fact objects representing only XBRL item facts which are direct children of this fact.

Fact.child_tuples
Returns a generator object of Fact objects representing only XBRL tuple facts which are direct children of this fact.

Fact.footnote_elements
Returns a generator object 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.select_footnotes(footnote_role = None, arc_role = None, link_role = None, lang = None)
Returns a generator object of xbrl.FootnoteResource objects for each footnote that has a concept-label relationship (http://www.xbrl.org/2003/arcrole/fact-footnote) to this fact and also matches the given parameters.

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`.

5.9.14 xbrl.FactSet

class **xbrl.FactSet**
A FactSet class represents a set of XBRL instance facts.

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

FactSet.copy()
Returns an xbrl.FactSet object which is an independent copy of the current fact set.

FactSet.merge(factSet)
Merges the facts represented by this object with the given factSet (equivalent to a set union).

FactSet.intersect(factSet)
Intersects the facts represented by this object with the given factSet (equivalent to a set intersection).

FactSet.filter(constraintSet)
Filters any facts that do not match the constraints specified in the constraintSet.

5.9.15 xbrl.FootnoteResource

class **xbrl.FootnoteResource(xbrl.Resource)**
The FootnoteResource class represents a footnote resource.

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

xbrl.FootnoteResource.lang
Returns the `xml:lang` attribute value as a string.

xbrl.FootnoteResource.text
Returns the text content of the footnote resource as a string.

5.9.16 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.

5.9.17 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 generator object of Context objects which represents all XBRL contexts present in the instance document.

Instance.units
Returns a generator object of Unit objects which represents all XBRL units present in the instance document.

Instance.facts
Returns a generator object 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 generator object 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 generator object 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.select_facts(constraintSet)
Returns an xbrl.FactSet representing the facts that match the constraints in the constraintSet.

Instance.tuples
Returns a generator object 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.

5.9.18 xbrl.LabelResource

class **xbrl.LabelResource(xbrl.Resource)**
The LabelResource class represents a label resource.

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

LabelResource.lang
Returns the xml:lang attribute value as a string.

LabelResource.text
Returns the text content of the label resource as a string.

5.9.19 xbrl.LayoutCell

class **xbrl.LayoutCell**
The LayoutCell class represents a cell in the generated table.

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

LayoutCell.row
Returns the table cell's Y coordinate.

LayoutCell.col
Returns the table cell's X coordinate.

LayoutCell.row_span
Returns the table cell's span in the vertical direction.

LayoutCell.col_span
Returns the table cell's span in the horizontal direction.

LayoutCell.is_span_start
Returns `True` if this table cell is the top left cell within a spanned region. Also returns `True` if the table cell doesn't have a span.

LayoutCell.is_header
Returns `True` if this table cell represents a header cell.

LayoutCell.is_data
Returns `True` if this table cell represents a data cell.

5.9.20 xbrl.LayoutDataCell

class **xbrl.LayoutDataCell(xbrl.LayoutCell)**
The LayoutDataCell class represents a data cell in the generated table.

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

LayoutDataCell.facts
Returns an xbrl.FactSet representing the facts that match the constraints of the cell. For under-specified tables, the returned fact set can contain multiple facts.

5.9.21 xbrl.LayoutHeaderCell

class **xbrl.LayoutHeaderCell(xbrl.LayoutCell)**
The LayoutHeaderCell class represents a header cell in the generated table.

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

LayoutHeaderCell.definition_node
Returns an xbrl.DefinitionNodeResource object representing the orignial definition node resource in the table linkbase.

LayoutHeaderCell.preferred_label_role
For header cells generated by a tree walk of the presentation linkbase, the preferredLabel value on the relationship leading to a given concept is returned. Otherwise None is returned.

LayoutHeaderCell.constraint_sets
Returns a generator object of tag to constraint set mappings. The untagged constraint set can be accessed with None as the key.

LayoutHeaderCell.tag_selectors
Returns a generator object of tag selectors (either directly specified or inherited).

LayoutHeaderCell.is_rollup
Returns True if the header cell represents a roll-up node (http://www.xbrl.org/specification/table-linkbase/pr-2013-12-18/table-linkbase-pr-2013-12-18.html#sec-roll-up-nodes).

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

LayoutHeaderCell.constraint_set(tag=None)
Returns an xbrl.ConstraintSet object for the given tag, or None if no constraint set with the given tag was found. If the tag is None, the untagged constraint set is returned.

5.9.22 xbrl.LayoutRow

class **xbrl.LayoutRow**
The LayoutRow class represents a row of the generated table.

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

LayoutRow.index
Returns the table row's coordinates.

LayoutRow.is_header_row
Returns `True` if this table row contains only header cells.

LayoutRow.cells
Returns a generator object of xbrl.LayoutHeaderCell and xbrl.LayoutDataCell objects representing the cells in the table row (header and data cells).

5.9.23 xbrl.LayoutTable

class **xbrl.LayoutTable**
The LayoutTable class represents the table after applying the resolution and layout process.

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

LayoutTable.tables
Returns an xbrl.TableResource object representing the original table resource in the table linkbase.

LayoutTable.rows
Returns a generator object of xbrl.LayoutRow objects representing the rows in the table (both header and data rows).

LayoutTable.width
Returns the total number of table columns (header and data columns).

LayoutTable.height
Returns the total number of table rows (header and data rows).

LayoutTable.header_width
Returns the number of header columns.

LayoutTable.header_height
Returns the number of header rows.

LayoutTable.data_width
Returns the number of data columns.

LayoutTable.data_height
Returns the number of data rows.

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

LayoutTable.cell(row, col)
Returns a xbrl.LayoutCell object representing the table cell in the row specified by `row` and the column specified by `col`.

5.9.24 xbrl.LayoutTableSet

class **xbrl.LayoutTableSet**
The LayoutTableSet class represents the table set after applying the resolution and layout process.

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

LayoutTableSet.tables
Returns a generator object of `xbrl.LayoutTable` objects.

5.9.25 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.aspect_value
Returns an xbrl.PeriodAspectValue object representing the value of the period aspect.

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`.

5.9.26 xbrl.PeriodAspectValue

class **xbrl.PeriodAspectValue**
The PeriodAspectValue class represents a value for the period aspect. The constructor takes the following arguments:

```
PeriodAspectValue( DTS dts, start = None, end = None )
```

The PeriodAspectValue class provides the following **constants**:

PeriodAspectValue.INSTANT
Denotes that the period represents a point in time.

PeriodAspectValue.DURATION
Denotes that the period represents a duration with a start point and end point.

PeriodAspectValue.FOREVER
Denotes that the period represents a duration without a start point or end point.

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

PeriodAspectValue.period_type
Returns `INSTANT`, `DURATION`, or `FOREVER`.

PeriodAspectValue.instant
If `period_type` is `INSTANT`, it returns a `datetime.datetime` object representing the single point in time. Otherwise it returns `None`.

PeriodAspectValue.start
If `period_type` is `DURATION`, it returns a `datetime.datetime` object representing the duration start point. Otherwise it returns `None`.

PeriodAspectValue.end
If `period_type` is `DURATION`, it returns a `datetime.datetime` object representing the duration end point. Otherwise it returns `None`.

5.9.27 xbrl.ReferencePart

class **xbrl.ReferencePart**
The ReferencePart class represents a reference part.

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

ReferencePart.element
Returns an xml.Element object which represents the XML element information item of the reference part.

ReferencePart.qname
Returns an xml.QName object which represents the XML qualified name of the reference part.

ReferencePart.text
Returns the text content of the reference part as a string.

5.9.28 xbrl.ReferenceResource

class **xbrl.ReferenceResource(xbrl.Resource)**
The ReferenceResource class represents a reference resource.

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

ReferenceResource.parts
Returns a generator object of xbrl.ReferencePart objects for each reference part.

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

ReferenceResource.find_part(qname)
Returns an xbrl.ReferencePart object if a reference part with the given xml.QName, qname, was found, otherwise None.

5.9.29 xbrl.Resource

class **xbrl.Resource**
The Resource class represents an XLink resource element within an extended link.

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

Resource.element
Returns an xml.Element object representing the XML element in the extended link.

Resource.id
Returns the id attribute value as a string, or None if the XML element doesn't have an id attribute.

Resource.role
Returns the xlink:role attribute value as a string, or None if the XML element doesn't have an xlink:role attribute.

Resource.labels
Returns a generator object of xbrl.LabelResource objects for each generic label that has an element-label (http://xbrl.org/arcrole/2008/element-label) relationship to this resource.

Resource.references
Returns a generator object of xbrl.ReferenceResource objects for each generic reference that has an element-reference (http://xbrl.org/arcrole/2008/element-reference) relationship to this resource.

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

Resource.select_labels(label_role = None, link_role = None, lang = None)
Returns a generator object of xbrl.LabelResource objects for each generic label that has an element-label relationship (http://xbrl.org/arcrole/2008/element-label) to this resource and also matches the given parameters.

Resource.select_references(reference_role = None, link_role = None)
Returns a generator object of xbrl.ReferenceResource objects for each generic reference that has an element-reference relationship (http://xbrl.org/arcrole/2008/element-reference) to this resource and also matches the given parameters.

5.9.30 xbrl.ScenarioAspectValue

class **xbrl.ScenarioAspectValue**

The ScenarioAspectValue class represents a value for the non-XDT scenario aspect. The constructor takes the following arguments:

```
ScenarioAspectValue( DTS dts, elements = None )
```

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

ScenarioAspectValue.elements

Returns a generator object of xml.Element objects representing the non-XDT child elements of the <scenario> XML element.

5.9.31 xbrl.SegmentAspectValue

class **xbrl.SegmentAspectValue**

The SegmentAspectValue class represents a value for the non-XDT segment aspect. The constructor takes the following arguments:

```
SegmentAspectValue( DTS dts, elements = None )
```

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

SegmentAspectValue.elements

Returns a generator object of xml.Element objects representing the non-XDT child elements of the XML element.

5.9.32 xbrl.TableError

class **xbrl.TableError(xbrl.Error)**
The xbrl.TableError class represents an error that is raised during the table resolution or layout process.

5.9.33 xbrl.TableResource

class **xbrl.TableResource(xbrl.Resource)**
The TableResource class represents a table resource in the table linkbase (http://www.xbrl.org/specification/table-linkbase/pr-2013-12-18/table-linkbase-pr-2013-12-18.html#sec-tables).

The TableResource class provides the following **instance method**:

TableResource.layout(instance)
Returns an xbrl.LayoutTableSet object representing the generated table set. If an error occurs during the resolution or layout process, an xbrl.TableError is raised.

5.9.34 xbrl.TypedDimensionAspectValue

class **xbrl.TypedDimensionAspectValue**
The TypedDimensionAspectValue class represents a value for the dimension aspect. The constructor takes the following arguments:

```
TypedDimensionAspectValue( DTS dts, dimension = None, value = None )
```

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

TypedDimensionAspectValue.is_explicit
Returns `True`.

TypedDimensionAspectValue.is_typed
Returns `False`.

TypedDimensionAspectValue.dimension
Returns an xbrl.Concept object representing the dimension.

TypedDimensionAspectValue.value
Returns an xml.Element object representing the dimension's typed domain value. The absent dimension aspect value is denoted by `None`.

5.9.35 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.aspect_value
Returns an xbrl.UnitAspectValue object representing the value of the unit aspect.

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.

5.9.36 xbrl.UnitAspectValue

class **xbrl.UnitAspectValue**

The UnitAspectValue class represents a value for the unit aspect. The constructor takes the following arguments:

```
UnitAspectValue (DTS dts, nominators = None, denominators = None )
```

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

UnitAspectValue.numerators

Returns a list of xml.QName objects which represents the QNames in the numerator of the XBRL unit.

UnitAspectValue.denominators

Returns a list of xml.QName objects which represents the QNames in the denominator of the XBRL unit.

Chapter 6

Java Interface

6 Java Interface

The RaptorXML API can be accessed from Java code. To access RaptorXML+XBRL Server from Java code, the libraries listed below must reside in the classpath. These libraries are installed in the `bin` folder of the installation folder.

- `RaptorXMLServer.jar`: The library that communicates with the RaptorXML server using HTTP requests
- `RaptorXMLServer_JavaDoc.zip`: A Javadoc file containing help documentation for the Java API

Note: In order to use the Java API, the Jar file must be on the Java Classpath. You may copy the Jar file to any location if this fits your project setup better than referencing it from the installed location.

Overview of the interface
The Java API is packaged in the `com.altova.raptorxml` package. The RaptorXML class provides an entry-point method called `getFactory()`, which returns `RaptorXMLFactory` objects. So, a `RaptorXMLFactory` instance can be created with the call: `RaptorXML.getFactory()`.

The `RaptorXMLFactory` interface provides methods for getting engine objects for validation and further processing (such as XSLT transformation).

Note: The `getFactory` method returns the respective factory object according to the RaptorXML edition installed.

The public interface of `RaptorXMLFactory` is described by the following listing:

```
public interface RaptorXMLFactory
{
   public XMLValidator getXMLValidator();
   public XBRL 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 setGlobalCatalog(String catalog);
   public void setUserCatalog(String catalog);
   public void setGlobalResourcesFile(String file);
   public void setGlobalResourceConfig(String config);
   public void setErrorFormat(ENUMErrorFormat format);
   public void setErrorLimit(int limit);
   public void setReportOptionalWarnings(boolean report);
```



```
}
```

For more details, see the descriptions of `RaptorXMLFactory` and the respective Java interfaces. Also see the Example Java Project.

6.1 Example Java Project

The Jave code listing below shows how basic functionality can be accessed. It is structured into the following parts:

- Locate the examples folder, and create a RaptorXML COM object instance
- Validate an XML file
- Perform an XSLT transformation, return the result as a string
- Process an XQuery document, return the result as a string
- Run the project

This basic functionality is included in the files in the `examples/API` folder of the RaptorXML+XBRL Server application folder.

```
public class RunRaptorXML
{
    // Locate samples installed with the product
    // (will be two levels higher from examples/API/Java)
    // REMARK: You might need to modify this path
       static final String strExamplesFolder = System.getProperty("user.dir") +
"/../../" ;

       static com.altova.raptorxml.RaptorXMLFactory rxml;
```

```
       static void ValidateXML() throws com.altova.raptorxml.RaptorXMLException
       {
           com.altova.raptorxml.XMLValidator xmlValidator =
rxml.getXMLValidator();
           System.out.println("RaptorXML Java - XML validation");
           xmlValidator.setInputXMLFromText( "<!DOCTYPE root [ <!ELEMENT root
(#PCDATA)> ]> <root>simple input document</root>" );
              if( xmlValidator.isWellFormed() )
                  System.out.println( "The input string is well-formed" );
              else
                  System.out.println( "Input string is not well-formed: " +
xmlValidator.getLastErrorMessage() );

              if( xmlValidator.isValid() )
                  System.out.println( "The input string is valid" );
              else
                  System.out.println( "Input string is not valid: " +
xmlValidator.getLastErrorMessage() );
       }
```

```
       static void RunXSLT() throws com.altova.raptorxml.RaptorXMLException
       {
```

```
            System.out.println("RaptorXML Java - XSL Transformation");
            com.altova.raptorxml.XSLT xsltEngine = rxml.getXSLT();
            xsltEngine.setInputXMLFileName( strExamplesFolder + "simple.xml" );
            xsltEngine.setXSLFileName( strExamplesFolder + "transform.xsl" );
            String result = xsltEngine.executeAndGetResultAsString();
            if( result == null )
                System.out.println( "Transformation failed: " +
xsltEngine.getLastErrorMessage() );
            else
                System.out.println( "Result is " + result );
        }
```

```
        static void RunXQuery() throws com.altova.raptorxml.RaptorXMLException
        {
            System.out.println("RaptorXML Java - XQuery execution");
            com.altova.raptorxml.XQuery xqEngine = rxml.getXQuery();
            xqEngine.setInputXMLFileName( strExamplesFolder + "simple.xml" );
            xqEngine.setXQueryFileName( strExamplesFolder + "CopyInput.xq" );
            System result = xqEngine.executeAndGetResultAsString();
            if( result == null )
                System.out.println( "Execution failed: " +
xqEngine.getLastErrorMessage() );
            else
                System.out.println( "Result is " + result );
        }
```

```
        public static void main(String[] args)
        {
            try
            {
                rxml = com.altova.raptorxml.RaptorXML.getFactory();
                rxml.setErrorLimit( 3 );

                ValidateXML();
                RunXSLT();
                RunXQuery();
            }

            catch( com.altova.raptorxml.RaptorXMLException e )
            {
                e.printStackTrace();
            }

        }

}
```

6.2 RaptorXML Interfaces for Java

Given below is a summary of the Java interfaces of the RaptorXML API. Detailed descriptions are given in the respective sections.

- `RaptorXMLFactory`
  Creates a new RaptorXML COM object instance via a native call, and provides access to RaptorXML engines.
- `XMLValidator`
  Interface for the XMLValidator Engine.
- `XSLT`
  Interface for the XSLT Engines.
- `XQuery`
  Interface for the XQuery Engines.
- `XBRL`
  Interface for the XBRL Engine.
- `RaptorXMLException`
  Interface for the `RaptorXMLException` method.

6.2.1 RaptorXMLFactory

`public interface` **`RaptorXMLFactory`**

Description

Use `RaptorXMLFactory()` to create a new RaptorXML COM object instance. This provides access to the RaptorXML engines. The relationship between `RaptorXMLFactory` and the RaptorXML COM object is one-to-one. This means that subsequent calls to the `get<ENGINENAME>()` function will return interfaces for the same engine instance.

The methods of the `RaptorXMLFactory` interface are described first, followed by its enumerations.

Methods

The methods of the class are described below in alphabetical order. In the table, they are organized into groups for ease of reference.

Engines	Errors, Warnings
getXBRL	setErrorFormat
getXMLValidator	setErrorLimit
getXQuery	setReportOptionalWarnings
getXSLT	

Catalogs	Global Resources	HTTP Server
setGlobalCatalog	setGlobalResourceConfig	setServerFile
setUserCatalog	setGlobalResourcesFile	setServerName
		setServerPort

Product Information	
getProductName	Is64Bit
getProductNameAndVersion	getAPIMajorVersion
getMajorVersion	getAPIMinorVersion
getMinorVersion	getAPIServicePackVersion
getServicePackVersion	

Top | Methods | Enumerations

`getAPIMajorVersion`

`public int` **`getAPIMajorVersion()`**
Returns the major version of the API as an integer. The major version of the API could be different from the product's major version if the API is connected to another server.
Returns:
an integer that is the major version of the API.

Top | Methods | Enumerations

`getAPIMinorVersion`
`public int` **`getAPIMinorVersion()`**
Returns the minor version of the API as an integer. The minor version of the API could be different from the product's minor version if the API is connected to another server.
Returns:
an integer that is the minor version of the API.

Top | Methods | Enumerations

`getAPIServicePackVersion`
`public int` **`getAPIServicePackVersion()`**
Returns the service pack version of the API as an integer. The service pack version of the API could be different from the product's service pack version if the API is connected to another server.
Returns:
an integer that is the service pack version of the API.

Top | Methods | Enumerations

`getMajorVersion`
`public int` **`getMajorVersion()`**
Returns the major version of the product as an integer. *Example:* For `Altova RaptorXML+XBRL Server 2014r2sp1(x64)`, returns `16` (the difference between the major version (`2014`) and the initial year `1998`). Throws a `RaptorXMLException` in case of an error.
Returns:
an integer that is the product's major version.

Top | Methods | Enumerations

`getMinorVersion`
`public int` **`getMinorVersion()`**
Returns the minor version of the product as an integer. *Example:* For `Altova RaptorXML+XBRL Server 2015r2sp1(x64)`, returns `2` (from the minor version number `r2`). Throws a `RaptorXMLException` in case of an error.
Returns:

an integer that is the product's minor version.

Top | Methods | Enumerations

getProductName

```
public string getProductName()
```

Returns the name of the product as a string. *Example:* For `Altova RaptorXML+XBRL Server 2015r2sp1(x64)`, returns `Altova RaptorXML+XBRL Server`. Throws a `RaptorXMLException` in case of an error.

Returns:

a string that is the product's name.

Top | Methods | Enumerations

getProductNameAndVersion

```
public string getProductNameAndVersion()
```

Returns the service pack version of the product as an integer. *Example:* For `Altova RaptorXML +XBRL Server 2015r2sp1(x64)`, returns `Altova RaptorXML+XBRL Server 2015r2sp1(x64)`. Throws a `RaptorXMLException` in case of an error.

Returns:

a string that is the product's name and version.

Top | Methods | Enumerations

getServicePackVersion

```
public int getServicePackVersion()
```

Returns the service pack version of the product as an integer. *Example:* For `RaptorXML+XBRL Server 2015r2sp1(x64)`, returns `1` (from the service pack version number `sp1`). Throws a `RaptorXMLException` in case of an error.

Returns:

an integer that is the product's service pack version.

Top | Methods | Enumerations

getXBRL

```
public XBRL getXBRL
```

Retrieves the XBRL engine.

Returns:

a new `XBRL` instance of this `RaptorXMLFactory`.

Top | Methods | Enumerations

getXMLValidator
public XMLValidator **getXMLValidator()**
Retrieves the XMLValidator.
Returns:
a new XMLValidator instance of this RaptorXMLFactory.

Top | Methods | Enumerations

getXQuery
public XQuery **getXQuery()**
Retrieves the XQuery engine.
Returns:
a new XQuery instance of this RaptorXMLFactory.

Top | Methods | Enumerations

getXSLT
public XSLT **getXSLT()**
Retrieves the XSLT engine.
Returns:
a new XSLT instance of this RaptorXMLFactory.

Top | Methods | Enumerations

is64Bit
public boolean **is64Bit()**
Checks if the application is a 64-bit executable. *Example:* For Altova RaptorXML+XBRL Server 2015r2sp1(x64), returns true. Throws a RaptorXMLException in case of an error.
Returns:
boolean true if the application is 64 bit, false if it is not.

Top | Methods | Enumerations

setErrorFormat
public void **setErrorFormat(**ENUMErrorFormat format**)**
Sets the RaptorXML error format to one of the ENUMErrorFormat literals (Text, ShortXML, LongXML).
Parameters:
format: Holds the value of the selected ENUMErrorFormat literal.

Top | Methods | Enumerations

setErrorLimit

```
public void setErrorLimit(int limit)
```

Sets the RaptorXML validation error limit.

Parameters:

`limit`: Is of type `int`, and specifies the number of errors to be reported before execution is halted. Use `-1` to set `limit` to be unlimited (that is, all errors will be reported). The default value is `100`.

Top | Methods | Enumerations

setGlobalCatalog

```
public void setGlobalCatalog(String catalog)
```

Sets the location, as a URL, of the main (entry-point) catalog file.

Parameters:

`catalog`: The supplied string must be an absolute URL that gives the exact location of the main catalog file to use.

Top | Methods | Enumerations

setGlobalResourceConfig

```
public void setGlobalResourceConfig(String config)
```

Sets the active configuration of the global resource.

Parameters:

`config`: Is of type `String`, and specifies the name of the configuration used by the active global resource.

Top | Methods | Enumerations

setGlobalResourcesFile

```
public void setGlobalResourcesFile(String file)
```

Sets the location, as a URL, of the Global Resources XML File.

Parameters:

`file`: The supplied string must be an absolute URL that gives the exact location of the Global Resources XML File.

Top | Methods | Enumerations

setReportOptionalWarnings

```
public void setReportOptionalWarnings(boolean report)
```

Enables/disables the reporting of warnings. A value of `true` enables warnings; `false` disables

them.
Parameters:
report: Takes boolean true or false.

Top | Methods | Enumerations

setServerFile
public void **setServerFile**(String file)
Sets the location of the HTTP server's configuration file relative to the HTTP server address. Raises a RaptorXMLException if an error occurs.
Parameters:
file: A string that gives the address of the HTTP server configuration file relative to the server address.

Top | Methods | Enumerations

setServerName
public void **setServerName**(String name)
Sets the name of the HTTP server. Raises a RaptorXMLException if an error occurs.
Parameters:
name: A string that gives the name of the HTTP server.

setServerPort
public void **setServerPort**(int port)
Sets the port on the HTTP server via which the service is accessed. The port must be fixed and known so that HTTP requests can be correctly addressed to the service. Raises a RaptorXMLException if an error occurs.
Parameters:
port: An integer that specifies the access port on the HTTP server.

Top | Methods | Enumerations

setUserCatalog
public void **setUserCatalog**(String catalog)
Sets the location, as a URL, of the custom user catalog file.
Parameters:
catalog: The supplied string must be an absolute URL that gives the exact location of the custom catalog file to use.

Top | Methods | Enumerations

Enumerations
ENUMErrorFormat

ENUMErrorFormat

```
public enum ENUMErrorFormat {
       eFormatText
       eFormatShortXML
       eFormatLongXML }
```

ENUMErrorFormat can take one of the enumeration literals: eFormatText, eFormatShortXML, eFormatLongXML. These set the format of the error messages, with eLongXML providing the most detailed messages. The default is eFormatText.

Used by (Interface::Method):

RaptorXMLFactory setErrorFormat

Top | Methods | Enumerations

6.2.2 XMLValidator

```
public interface XMLValidator
```

Description
Validates the supplied XML document, schema document, or DTD document. XML document validation can be done with internal or external DTDs or XML Schemas. Also checks the well-formedness of XML, DTD, and XML Schema documents. The methods of the interface are described first, followed by its enumerations.

Methods
The methods of the class are described below in alphabetical order. In the table, they are organized into groups for ease of reference.

Processing	Input Files	XML Schema
isValid(ENUM type)	setInputXMLFileName	setSchemaImports
isValid	setInputXMLFromText	setSchemalocationHints
isWellFormed(ENUM type)	setInputXMLFileCollection	setSchemaMapping
isWellFormed	setInputXMLTextCollection	setXSDVersion
getLastErrorMessage	setSchemaFileName	
setAssessmentMode	setSchemaFromText	**XML**
setPythonScriptFile	setSchemaFileCollection	setEnableNamespaces
setStreaming	setSchemaTextCollection	setXincludeSupport
	setDTDFileName	setXMLValidationMode
	setDTDFromText	

Top | Methods | Enumerations

getLastErrorMessage
`public String getLastErrorMessage()`
Retrieves the last error message from the XML Validator engine.
Returns:
a string that is the last error message from the XML Validator engine.

Top | Methods | Enumerations

isValid

public boolean **isValid(**ENUMValidationType type**)**
Returns the result of validating the XML document, schema document, or DTD document. The type of document to validate is specified by the type parameter, which takes an ENUMValidationType literal as its value. The result is true on success, false on failure. If an error occurs, a RaptorXMLException is raised. Use the getLastErrorMessage method to access additional information.
Parameters:
type: An ENUMValidationType literal, which specifies whether the validation is of an XML Schema, or of a DTD, or of an XML document against an XML Schema, or of an XML document against a DTD.
Returns:
boolean true on success, false on failure.

Top | Methods | Enumerations

isValid
public boolean **isValid()**
Returns the result of validating the submitted document.The result is true on success, false on failure.
Returns:
boolean true on success, false on failure.

Top | Methods | Enumerations

isWellFormed
public boolean **isWellFormed(**ENUMWellformedCheckType type**)**
Returns the result of checking the XML document or DTD document for well-formedness. The type of document to check is specified by the type parameter, which takes an ENUMWellformedCheckType literal as its value. The result is true on success, false on failure. If an error occurs, a RaptorXMLException is raised. Use the getLastErrorMessage method to access additional information.
Parameters:
type: An ENUMWellformedCheckType literal, which specifies whether an XML document or DTD will be checked for well-formedness.
Returns:
boolean true on success, false on failure.

Top | Methods | Enumerations

isWellFormed
public boolean **isWellFormed()**
Returns the result of checking the XML document or DTD document for well-formedness. The result is true on success, false on failure.
Returns:
boolean true on success, false on failure.

Top | Methods | Enumerations

setAssessmentMode

```
public void setAssessmentMode(ENUMAssessmentMode mode)
```

Sets the assessment mode of the XML validation (`Strict/Lax`), which is defined in the `mode` parameter that takes an `ENUMAssessmentMode` literal.

Parameters:

`mode`: An `ENUMAssessmentMode` literal, which specifies whether the validation should be strict or lax or should be skipped.

Top | Methods | Enumerations

setDTDFileName

```
public void setDTDFileName(String filePath)
```

Sets the location, as a URL, of the DTD document to use for validation.

Parameters:

`filePath`: The supplied string must be an absolute URL that gives the exact location of the DTD file to use.

Top | Methods | Enumerations

setDTDFromText

```
public void setDTDFromText(String dtdText)
```

Supplies the content of the DTD document as text.

Parameters:

`dtdText`: The supplied string is the DTD document to be used for validation.

Top | Methods | Enumerations

setEnableNamespaces

```
public void setEnableNamespaces(boolean enable)
```

Enables namespace-aware processing. This is useful for checking the XML instance for errors due to incorrect namespaces. A value of `true` enables namespace-aware processing; `false` disables it. Default is `false`.

Parameters:

`support`: Takes boolean `true` or `false`.

Top | Methods | Enumerations

setInputXMLFileCollection

```
public void setInputXMLFileCollection(Collection<?> fileCollection)
```

Supplies the collection of XML files that will be used as input data. The files are identified by their URLs.

Parameters:

`fileCollection`: A collection of strings, each of which is the absolute URL of an input XML file.

Top | Methods | Enumerations

setInputXMLFileName

```
public void setInputXMLFileName(String filePath)
```

Sets the location, as a URL, of the XML document to be validated.

Parameters:

`filePath`: The supplied string must be an absolute URL that gives the exact location of the XML file.

Top | Methods | Enumerations

setInputXMLFromText

```
public void setInputXMLFromText(String inputText)
```

Supplies the contents of the XML document to validate.

Parameters:

`inputText`: The supplied string is the content of the XML document to validate.

Top | Methods | Enumerations

setInputXMLTextCollection

```
public void setInputXMLTextCollection(Collection<?> stringCollection)
```

Supplies the content of multiple XML files that will be used as input data.

Parameters:

`stringCollection`: A collection of strings, each of which is the content of an input XML file.

Top | Methods | Enumerations

setParallelAssessment

```
public void setParallelAssessment(boolean support)
```

Enables or disables the use of parallel assessment. A value of `true` enables parallel asessment; `false` disables it. The default value is `false`.

Parameters:

`support`: Takes boolean `true` or `false`.

Top | Methods | Enumerations

setPythonScriptFile

```
public void setPythonScriptFile(String file)
```

Sets the location, as a URL, of the Python script file.

Parameters:

`file`: The supplied string must be an absolute URL that gives the exact location of the Python file.

Top | Methods | Enumerations

setSchemaFileCollection

```
public void setSchemaFileCollection(Collection<?> fileCollection)
```

Supplies the collection of XML files that will be used as external XML Schemas. The files are identified by their URLs.

Parameters:

`fileCollection`: A collection of strings, each of which is the absolute URL of an XML Schema file.

Top | Methods | Enumerations

setSchemaFileName

```
public void setSchemaFileName(String filePath)
```

Sets the location, as a URL, of the XML Schema document to be used.

Parameters:

`filePath`: The supplied string must be an absolute URL that gives the exact location of the XML Schema file.

Top | Methods | Enumerations

setSchemaFromText

```
public void setSchemaFromText(String schemaText)
```

Supplies the contents of the XML Schema document to use.

Parameters:

`schemaText`: The supplied string is the content of the XML Schema document to use.

Top | Methods | Enumerations

setSchemaImports

```
public void setSchemaImports(ENUMSchemaImports opt)
```

Specifies how schema imports are to be handled based on the attribute values of the `xs:import`

elements. The kind of handling is specified by the ENUMSchemaImports literal that is selected.
Parameters:
opt: Holds the ENUMSchemaImports literal, which determines the handling of schema imports. See the description of ENUMSchemaImports for details.

Top | Methods | Enumerations

setSchemalocationHints
public void **setSchemalocationHints**(ENUMLoadSchemalocation opt)
Specifies the mechanism to use to locate the schema. The mechanism is specified by the ENUMLoadSchemalocation literal that is selected.
Parameters:
opt: Holds the ENUMLoadSchemalocation literal, which determines which schema location mechanism to use. See the description of ENUMLoadSchemalocation for details.

Top | Methods | Enumerations

setSchemaMapping
public void **setSchemaMapping**(ENUMSchemaMapping opt)
Sets what mapping to use in order to locate the schema. The mapping is specified by the ENUMSchemaMapping literal that is selected.
Parameters:
opt: Holds the ENUMSchemaMapping literal. See the description of ENUMSchemaMapping for details.

Top | Methods | Enumerations

setInputSchemaTextCollection
public void **setInputSchemaTextCollection**(Collection<?> stringCollection)
Supplies the content of multiple XML Schema documents.
Parameters:
stringCollection: A collection of strings, each of which is the content of an XML Schema document.

Top | Methods | Enumerations

setStreaming
public void **setStreaming**(boolean support)
Enables streaming validation. In streaming mode, data stored in memory is minimized and processing is faster.
Parameters:
support: A value of true enables streaming; false disables it. Default is true.

Top | Methods | Enumerations

setXincludeSupport

```
public void setXIncludeSupport(boolean support)
```

Enables or disables the use of `XInclude` elements. A value of `true` enables XInclude support; `false` disables it. The default value is `false`.

Parameters:

`support`: Takes boolean `true` or `false`.

Top | Methods | Enumerations

setXMLValidationMode

```
public void setXMLValidationMode(ENUMXMLValidationMode mode)
```

Sets the XML validation mode, which is an enumeration literal of `ENUMXMLValidationMode`.

Parameters:

`mode`: Is an enumeration literal of `ENUMXMLValidationMode` that determines whether to check validity or well-formedness.

Top | Methods | Enumerations

setXSDVersion

```
public void setXSDVersion(ENUMXSDVersion version)
```

Sets the XML Schema version against which the XML document will be validated.

Parameters:

`version`: Is an enumeration literal of `ENUMXSDVersion` that sets the XML Schema version.

Top | Methods | Enumerations

Enumerations
ENUMAssessmentMode
ENUMLoadSchemalocation
ENUMSchemaImports
ENUMSchemaMapping
ENUMXMLValidationMode
ENUMValidationType
ENUMWellformedCheckType
ENUMXSDVersion

Top | Methods | Enumerations

ENUMAssessmentMode

```
public enum ENUMAssessmentMode {
       eAssessmentModeLax
       eAssessmentModeStrict   }
```

ENUMAssessmentMode takes one of the enumeration literals: eAssessmentModeLax, eAssessmentModeStrict. These set whether validation should be lax or strict.

Used by (Interface::Method):

XMLValidator	setAssessmentMode

Top | Methods | Enumerations

ENUMLoadSchemalocation

```
public enum ENUMLoadSchemalocation {
       eLoadBySchemalocation
       eLoadByNamespace
       eLoadCombiningBoth
       eLoadIgnore   }
```

ENUMLoadSchemalocation contains the enumeration literal that specifies the schema locating mechanism. The selection is based on the schema location attribute of the XML or XBRL instance document. This attribute could be xsi:schemaLocation or xsi:noNamespaceSchemaLocation.

- eLoadBySchemalocation uses the URL of the schema location attribute in the XML or XBRL instance document. This enumeration literal is the **default value**.
- eLoadByNamespace uses the namespace part of xsi:schemaLocation and an empty string in the case of xsi:noNamespaceSchemaLocation to locate the schema via a catalog mapping.
- eLoadCombiningBoth: If either the namespace URL or schema location URL has a catalog mapping, then the catalog mapping is used. If both have catalog mappings, then the value of ENUMSchemaMapping decides which mapping is used. If neither the namespace nor schema location has a catalog mapping, the schema location URL is used.
- eLoadCombiningBoth: The xsi:schemaLocation and xsi:noNamespaceSchemaLocation attributes are both ignored.

Used by (Interface::Method):

XMLValidator	setSchemalocationHints
XSLT	setSchemalocationHints
XBRL	setSchemalocationHints

Top | Methods | Enumerations

ENUMSchemaImports

```
public enum ENUMSchemaImports {
       eSILoadBySchemalocation
       eSILoadPreferringSchemalocation
       eSILoadByNamespace
       eSILoadCombiningBoth
       eSILicenseNamespaceOnly   }
```

ENUMSchemaImports contains the enumeration literal that defines the behavior of the schema's xs:import elements, each of which has an optional namespace attribute and an optional schemaLocation attribute.

- eSILoadBySchemalocation uses the value of the schemaLocation attribute to locate the schema, taking account of catalog mappings. If the namespace attribute is present, the namespace is imported (licensed).
- eSILoadPreferringSchemalocation: 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 enumeration literal is the **default value**.
- eSILoadByNamespace uses the value of the namespace attribute to locate the schema via a catalog mapping.
- eSILoadCombiningBoth: If either the namespace URL or schemaLocation URL has a catalog mapping, then the catalog mapping is used. If both have catalog mappings, then the value of ENUMSchemaMapping decides which mapping is used. If neither the namespace nor schemaLocation URL has a catalog mapping, the schemaLocation URL is used.
- eSILicenseNamespaceOnly: The namespace is imported. No schema document is imported.

Used by (Interface::Method):

XMLValidator	setSchemaImports
XSLT	setSchemaImports
XBRL	setSchemaImports

Top | Methods | Enumerations

ENUMSchemaMapping

```
public enum ENUMSchemaMapping {
       eSMPreferSchemalocation
       eSMPreferNamespace   }
```

ENUMSchemaMapping contains the enumeration literal that specifies whether the namespace or the schema-location is to be selected.

- eSMPreferNamespace: Selects the namespace.
- eSMPreferSchemalocation: Selects the schema location. This is the default value.

Used by (Interface::Method):

XMLValidator	setSchemaMapping
XSLT	setSchemaMapping

XBRL	setSchemaMapping

Top | Methods | Enumerations

ENUMXMLValidationMode

```
public enum ENUMXMLValidationMode {
       eProcessingModeValid
       eProcessingModeWF  }
```

ENUMXMLValidationMode contains the enumeration literal specifying the type of XML validation to perform (validation or well-formedness check).

- eProcessingModeValid: Sets the XML processing mode to validation.
- eProcessingModeValid: Sets the XML processing mode to wellformed. This is the default value.

Used by (Interface::Method):

XMLValidator	setXMLValidationMode
XSLT	setXMLValidationMode
XQuery	setXMLValidationMode

Top | Methods | Enumerations

ENUMValidationType

```
public enum ENUMValidationType {
       eValidateAny
       eValidateXMLWithDTD
       eValidateXMLWithXSD
       eValidateDTD
       eValidateXSD }
```

ENUMValidationType contains the enumeration literal specifying what validation to carry out and, in the case of XML documents, whether validation is against a DTD or XSD.

- eValidateAny: The document type is detected automatically.
- eValidateXMLWithDTD: Validates an XML document against a DTD.
- eValidateXMLWithXSD: Validates an XML document against an XSD (XML Schema).
- eValidateDTD: Validates a DTD document.
- eValidateXSD: Validates an XSD document.

Used by (Interface::Method):

XMLValidator	isValid

Top | Methods | Enumerations

ENUMWellformedCheckType

```
public enum ENUMWellformedCheckType {
       eWellformedAny
       eWellformedXML
       eWellformedDTD   }
```

ENUMWellformedCheckType contains the enumeration literal specifying the type of well-formed check to make (for XML or DTD documents).

- eWellformedAny: The document type is detected automatically.
- eWellformedXML: Checks an XML document for well-formedness.
- eWellformedDTD: Checks a DTD document for well-formedness.

Used by (Interface::Method):

XMLValidator	isWellformed

Top | Methods | Enumerations

ENUMXSDVersion

```
public enum ENUMXSDVersion {
       eXSDVersionAuto
       eXSDVersion10
       eXSDVersion11   }
```

ENUMXSDVersion contains the enumeration literal specifying the XML Schema version.

- eXSDVersionAuto: The XML Schema version is detected automatically from the XSD document's vc:minVersion attribute. If the XSD document's vc:minVersion attribute has a value of 1.1, the document will be considered to be XSD 1.1. If the attribute has any other value, or does not exist, the document will be considered to be XSD 1.0.
- eXSDVersion10: Sets the XML Schema version for validation to XML Schema 1.0.
- eXSDVersion11: Sets the XML Schema version for validation to XML Schema 1.1.

Used by (Interface::Method):

XMLValidator	setXSDVersion
XSLT	setXSDVersion
XQuery	setXSDVersion

Top | Methods | Enumerations

6.2.3 XSLT

```
public interface XSLT
```

Description
Transforms XML using the supplied XSLT 1.0, 2.0, or 3.0 document. XML and XSLT documents can be provided as files (via a URL) or as a text string. Output is returned as a file (at a named location) or as a text string. XSLT parameters can be supplied, and Altova extension functions can be enabled for specialized processing, such as for charts. The XSLT document can also be validated. Where string inputs are to be interpreted as URLs, absolute paths should be used.

The methods of the XSLT interface are described first, followed by its enumerations.

Methods
The methods of the class are described below in alphabetical order. In the table, they are organized into groups for ease of reference.

Processing	XSLT
isValid	setVersion
execute	setXSLFileName
executeAndGetResultAsString	setXSLFromText
executeAndGetResultAsStringWithBaseOutputURI	setaddExternalParameter
getLastErrorMessage	setclearExternalParametersList
setIndentCharacters	setInitialTemplateMode
setStreamingSerialization	setNamedTemplateEntryPoint

XML Schema	XML	Extensions
setSchemaImports	setInputXMLFileName	setChartExtensionsEnabled
setSchemalocationHints	setInputXMLFromText	setDotNetExtensionsEnabled
setSchemaMapping	setLoadXMLWithPSVI	setJavaExtensionsEnabled
setXSDVersion	setXincludeSupport	setJavaBarcodeExtensionLocation
	setXMLValidationMode	

Top | Methods | Enumerations

addExternalParameter

```
public void addExternalParameter(String name, String value)
```

Adds the name and value of a new external parameter. Each external parameter and its value is to

be specified in a separate call to the method. Parameters must be declared in the XSLT document. Since parameter values are XPath expressions, parameter values that are strings must be enclosed in single quotes.
Parameters:
`name`: Holds the name of the parameter, which is a QName, as a string.
`value`: Holds the value of the parameter as a string.

Top | Methods | Enumerations

clearExternalParameterList
`public void` **clearExternalVariableList()**
Clears the external parameters list created by the `AddExternalParameter` method.

Top | Methods | Enumerations

execute
`public boolean` **execute(**`String outputFile`**)**
Executes the XSLT transformation according to the XSLT specification named in `ENUMXSLTVersion` (see the `setVersion` method), and saves the result to the output file named in the `outputFile` parameter. If an error occurs, a `RaptorXMLException` is raised. Use the `getLastErrorMessage` method to access additional information.
Parameters:
`outputFile`: A string that provides the location (path and filename) of the output file.
Returns:
boolean `true` on successful execution, `false` on failure.

Top | Methods | Enumerations

executeAndGetResultAsString
`public String` **executeAndGetResultAsString()**
Executes the XSLT transformation according to the XSLT specification named in `ENUMXSLTVersion` (see the `setVersion` method), and returns the result as a string. If an error occurs, a `RaptorXMLException` is raised. Use the `getLastErrorMessage` method to access additional information.
Returns:
a string that is the result of the XSLT transformation.

Top | Methods | Enumerations

executeAndGetResultAsStringWithBaseOutputURI
`public String` **executeAndGetResultAsStringWithBaseOutputURI(**`String baseURI`**)**
Executes the XSLT transformation according to the XSLT specification named in

ENUMXSLTVersion (see the setVersion method), and returns the result as a string at the location defined by the base URI.. If an error occurs, a RaptorXMLException is raised. Use the getLastErrorMessage method to access additional information.
Parameters:
baseURI: A string that provides a URI.
Returns:
a string that is the result of the XSLT transformation.

Top | Methods | Enumerations

getLastErrorMessage
public String **getLastErrorMessage()**
Retrieves the last error message from the XSLT engine.
Returns:
a string that is the last error message from the XSLT engine.

Top | Methods | Enumerations

isValid
public boolean **isValid()**
Returns the result of validating the XSLT document according to the XSLT specification named in ENUMXSLTVersion (see the setVersion method). The result is true on success, false on failure. If an error occurs, a RaptorXMLException is raised. Use the getLastErrorMessage method to access additional information.
Returns:
boolean true on success, false on failure.

Top | Methods | Enumerations

setChartExtensionsEnabled
public void **setChartExtensionsEnabled(**boolean enable**)**
Enables or disables Altova's chart extension functions.
Parameters:
enable: A value of true enables chart extensions; false disables them. Default value is true.

Top | Methods | Enumerations

setDotNetExtensionsEnabled
public void **setDotNetExtensionsEnabled(**boolean enable**)**
Enables or disables .NET extension functions.
Parameters:
enable: A value of true enables .NET extensions; false disables them. Default value is true.

Top | Methods | Enumerations

setJavaBarcodeExtensionLocation

```
public void setJavaBarcodeExtensionLocation(String path)
```

Specifies the location of the barcode extension file. See the section on Altova's barcode extension functions for more information.

Parameters:

`path`: The supplied string must be an absolute URL that gives the base location of the file to use.

Top | Methods | Enumerations

setJavaExtensionsEnabled

```
public void setJavaExtensionsEnabled(boolean enable)
```

Enables or disables Java extension functions.

Parameters:

`enable`: A value of `true` enables Java extensions; `false` disables them. Default value is `true`.

Top | Methods | Enumerations

setIndentCharacters

```
public void setIndentCharacters(String chars)
```

Sets the character string that will be used as indentation in the output.

Parameters:

`chars`: Holds the indentation character string.

Top | Methods | Enumerations

setInitialTemplateMode

```
public void setInitialTemplateMode(String mode)
```

Sets the name of the initial template mode. Processing will start with templates having this mode value. Transformation must be started after assigning the XML and XSLT documents.

Parameters:

`mode`: The name of the initial template mode, as a string.

Top | Methods | Enumerations

setInputXMLFileName

```
public void setInputXMLFileName(String xmlFile)
```

Sets the location, as a URL, of the XML document to be transformed.

Parameters:
`xmlFile`: The supplied string must be an absolute URL that gives the exact location of the XML file to use.

Top | Methods | Enumerations

setInputXMLFromText
`public void setInputXMLFromText(String xmlText)`
Supplies the contents of the XML input document as text.
Parameters:
`xmlText`: The supplied string is the XML data to be processed.

Top | Methods | Enumerations

setLoadXMLWithPSVI
`public void setLoadXMLWithPSVI(boolean load)`
Enables or disables the option to load and use the Post Schema Validation Infoset (PSVI). If the PSVI is loaded, information obtained from the schema can be used to qualify data in the XML document. A value of `true` enables PSVI loading; `false` disables it.
Parameters:
`load`: Takes boolean `true` or `false`.

Top | Methods | Enumerations

setNamedTemplateEntryPoint
`public void setNamedTemplateEntryPoint(String template)`
Gives the name of the named template with which processing is to start.
Parameters:
`template`: The name of the named template, as a string.

Top | Methods | Enumerations

setSchemaImports
`public void setSchemaImports(ENUMSchemaImports opt)`
Specifies how schema imports are to be handled based on the attribute values of the `xs:import` elements. The kind of handling is specified by the `ENUMSchemaImports` literal that is selected.
Parameters:
`opt`: Holds the `ENUMSchemaImports` literal, which determines the handling of schema imports.

Top | Methods | Enumerations

setSchemalocationHints

```
public void setSchemalocationHints(ENUMLoadSchemalocation opt)
```

Specifies the mechanism to use to locate the schema. The mechanism is specified by the ENUMLoadSchemalocation literal that is selected.

Parameters:

opt: Holds the ENUMLoadSchemalocation literal, which determines which schema location mechanism to use.

Top | Methods | Enumerations

setSchemaMapping

```
public void setSchemaMapping(ENUMSchemaMapping opt)
```

Sets what mapping to use in order to locate the schema. The mapping is specified by the ENUMSchemaMapping literal that is selected.

Parameters:

opt: Holds the ENUMSchemaMapping literal.

Top | Methods | Enumerations

setStreamingSerialization

```
public void setStreamingSerialization(boolean support)
```

Enables streaming serialization. In streaming mode, data stored in memory is minimized and processing is faster.

Parameters:

support: A value of true enables streaming serialization; false disables it.

Top | Methods | Enumerations

setVersion

```
public void setVersion(EnumXSLTVersion version)
```

Sets the XSLT version to use for processing (validation or XSLT transformation).

Parameters:

version: Holds an EnumXSLTVersion enumeration literal eVersion10, eVersion20, or eVersion30.

Top | Methods | Enumerations

setXincludeSupport

```
public void setXIncludeSupport(boolean support)
```

Enables or disables the use of XInclude elements. A value of true enables XInclude support; false disables it. The default value is false.

Parameters:
support: Takes boolean true or false.

Top | Methods | Enumerations

setXMLValidationMode
public void **setXMLValidationMode**(ENUMXMLValidationMode mode)
Sets the XML validation mode, which is an enumeration literal of ENUMXMLValidationMode.
Parameters:
mode: Is an enumeration literal of ENUMXMLValidationMode.

Top | Methods | Enumerations

setXSDVersion
public void **setXSDVersion**(ENUMXSDVersion version)
Sets the XML Schema version against which the XML document will be validated.
Parameters:
version: Is an enumeration literal of ENUMXSDVersion.

Top | Methods | Enumerations

setXSLFileName
public void **setXSLFileName**(String xslFile)
Sets the location, as a URL, of the XSLT document to be used for the transformation.
Parameters:
xslFile: The supplied string must be an absolute URL that gives the exact location of the XSLT file.

Top | Methods | Enumerations

setXSLFromText
public void **setXSLFromText**(String xslText)
Supplies the contents of the XSLT document as text.
Parameters:
xslText: The supplied string is the XSLT document to be used for the transformation.

Top | Methods | Enumerations

Enumerations
ENUMXSLTVersion

ENUMXSLTVersion

```
public enum ENUMXSLTVersion {
       eVersion10
       eVersion20
       eVersion30 }
```

ENUMXSLTVersion takes one of the enumeration literals: eVersion10, eVersion20, eVersion30. These set the XSLT version to be used for processing (validation or XSLT transformation).

Used by (Interface::Method):

XSLT	setVersion

Top | Methods | Enumerations

6.2.4 XQuery

```
public interface XQuery
```

Description
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 as a text string. Output is returned as a file (at a named location) or as a text string. External XQuery variables can be supplied, and a number of serialization options are available. The XQuery document can also be validated. Where string inputs are to be interpreted as URLs, absolute paths should be used.

The methods of the XQuery interface are described first, followed by its enumerations.

Methods
The methods of the class are described below in alphabetical order. In the table, they are organized into groups for ease of reference.

Processing	XML	XQuery
isValid	setInputXMLFileName	setVersion
isValidUpdate	setInputXMLFromText	setXQueryFileName
execute	setLoadXMLWithPSVI	setXQueryFromText
executeAndGetResultAsString	setXincludeSupport	addExternalVariable
executeUpdate	setXMLValidationMode	clearExternalVariableList
executeUpdateAndGetResultAsString	setXSDVersion	
getLastErrorMessage		
setUpdatedXMLWriteMode		

Serialization Options	Extensions
setIndentCharacters	setChartExtensionsEnabled
setKeepFormatting	setDotNetExtensionsEnabled
setOutputEncoding	setJavaExtensionsEnabled
setOutputIndent	
setOutputMethod	
setOutputOmitXMLDeclaration	

Top | Methods | Enumerations

addExternalVariable

```
public void addExternalVariable(String name, String value)
```

Adds the name and value of a new external variable. 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 (with an optional type declaration). If the variable value is a string, enclose the value in single quotes.

Parameters:

`name`: Holds the name of the variable, which is a QName, as a string.

`value`: Holds the value of the variable as a string.

Top | Methods | Enumerations

clearExternalVariableList

```
public void clearExternalVariableList()
```

Clears the external variables list created by the `AddExternalVariable` method.

Top | Methods | Enumerations

execute

```
public boolean execute(String outputFile)
```

Executes the XQuery transformation according to the XQuery specification named in `ENUMXQueryVersion` (see the `setVersion` method), and saves the result to the output file named in the `outputFile` parameter.

Parameters:

`outputFile`: A string that provides the location (path and filename) of the output file.

Returns:

boolean `true` on successful execution, `false` on failure.

Top | Methods | Enumerations

executeUpdate

```
public boolean executeUpdate(String outputFile)
```

Executes the XQuery update according to the XQuery Update specification named in `ENUMXQueryVersion` (see the `setVersion` method), and saves the result to the output file named in the `outputFile` parameter.

Parameters:

`outputFile`: A string that provides the location (path and filename) of the output file.

Returns:

boolean `true` on successful execution, `false` on failure.

Top | Methods | Enumerations

executeAndGetResultAsString

public String **executeAndGetResultAsString()**

Executes the XQuery transformation according to the XQuery specification named in ENUMXQueryVersion (see the setVersion method), and returns the result as a string.

Returns:

a string that is the result of the XQuery execution.

Top | Methods | Enumerations

executeUpdateAndGetResultAsString

public String **executeUpdateAndGetResultAsString()**

Executes the XQuery update according to the XQuery Update specification named in ENUMXQueryVersion (see the setVersion method), and returns the result as a string.

Returns:

a string that is the result of the XQuery update.

Top | Methods | Enumerations

getLastErrorMessage

public String **getLastErrorMessage()**

Retrieves the last error message from the XQuery engine.

Returns:

a string that is the last error message from the XQuery engine.

Top | Methods | Enumerations

isValid

public boolean **isValid()**

Returns the result of validating the XQuery document according to the XQuery specification named in ENUMXQueryVersion (see the setVersion method). The result is true on success, false on failure. If an error occurs, a RaptorXMLException is raised. Use the getLastErrorMessage method to access additional information.

Returns:

boolean true on success, false on failure.

Top | Methods | Enumerations

isValidUpdate

public boolean **isValidUpdate()**

Returns the result of validating the XQuery Update document according to the XQuery Update specification named in ENUMXQueryVersion (see the setVersion method). The result is true on

success, `false` on failure. If an error occurs, a `RaptorXMLException` is raised. Use the `getLastErrorMessage` method to access additional information.
Returns:
boolean `true` on success, `false` on failure.

Top | Methods | Enumerations

setChartExtensionsEnabled
`public void setChartExtensionsEnabled(boolean enable)`
Enables or disables Altova's chart extension functions.
Parameters:
`enable`: A value of `true` enables chart extensions; `false` disables them. Default value is `true`.

Top | Methods | Enumerations

setDotNetExtensionsEnabled
`public void setDotNetExtensionsEnabled(boolean enable)`
Enables or disables .NET extension functions.
Parameters:
`enable`: A value of `true` enables .NET extensions; `false` disables them. Default value is `true`.

Top | Methods | Enumerations

setIndentCharacters
`public void setIndentCharacters(String chars)`
Sets the character string that will be used as indentation in the output.
Parameters:
`chars`: Holds the indentation character string.

Top | Methods | Enumerations

setInputXMLFileName
`public void setInputXMLFileName(String xmlFile)`
Sets the location, as a URL, of the XML document to be used for the XQuery execution.
Parameters:
`xmlFile`: The supplied string must be an absolute URL that gives the exact location of the XML file to use.

Top | Methods | Enumerations

setInputXMLFromText

```
public void setInputXMLFromText(String xmlText)
```

Supplies the contents of the XML input document as text.

Parameters:

`xmlText`: The supplied string is the XML data to be processed.

Top | Methods | Enumerations

setJavaExtensionsEnabled

```
public void setJavaExtensionsEnabled(boolean enable)
```

Enables or disables Java extension functions.

Parameters:

`enable`: A value of `true` enables Java extensions; `false` disables them. Default value is `true`.

Top | Methods | Enumerations

setKeepFormatting

```
public void setKeepFormatting(boolean keep)
```

Enables or disables the option to keep the original formatting of files that will be updated by `executeUpdate`.

Parameters:

`keep`: Takes boolean `true` or `false`.

Top | Methods | Enumerations

setLoadXMLWithPSVI

```
public void setLoadXMLWithPSVI(boolean load)
```

Enables or disables the option to load and use the Post Schema Validation Infoset (PSVI). If the PSVI is loaded, information obtained from the schema can be used to qualify data in the XML document. A value of `true` enables PSVI loading; `false` disables it.

Parameters:

`load`: Takes boolean `true` or `false`.

Top | Methods | Enumerations

setOutputEncoding

```
public void setOutputEncoding(String encoding)
```

Sets the encoding of the result document.

Parameters:

`encoding`: Use an official IANA encoding name, such as `UTF-8`, `UTF-16`, `US-ASCII`, `ISO-8859-1`, as a string.

Top | Methods | Enumerations

setOutputIndent
`public void` **setOutputIndent(**`boolean indent`**)**
Enables or disables indentation in the output document.
Parameters:
`indent`: A value of `true` enables indentation; `false` disables it.

Top | Methods | Enumerations

setOutputMethod
`public void` **setOutputMethod(**`String outputMethod`**)**
Specifies the serialization of the output document.
Parameters:
`outputMethod`: Valid values are: `xml | xhtml | html | text`, given as a string. Default value is `xml`.

Top | Methods | Enumerations

setOutputOmitXMLDeclaration
`public void` **setOutputOmitXMLDeclaration(**`boolean omit`**)**
Enables or disables the inclusion of the XML declaration in the result document.
Parameters:
`omit`: A value of `true` omits the declaration; `false` includes it. Default value is `false`.

Top | Methods | Enumerations

setUpdatedXMLWriteMode
`public void` **setUpdatedXMLWriteMode(**`EnumXQueryUpdatedXML mode`**)**
Sets the mode to use for updating.
Parameters:
`mode`: Holds an `EnumXQueryUpdatedXML` enumeration literal `eUpdatedDiscard`, `eUpdatedWriteback` or `eUpdatedAsMainResult`.

Top | Methods | Enumerations

setVersion
`public void` **setVersion(**`EnumXQueryVersion version`**)**
Sets the XQuery version to use for processing (validation or XQuery execution).
Parameters:

version: Holds an EnumXQueryVersion enumeration literal eVersion10 or eVersion30. Default is eVersion30ml.

Top | Methods | Enumerations

setXincludeSupport

```
public void setXIncludeSupport(boolean support)
```

Enables or disables the use of XInclude elements. A value of true enables XInclude support; false disables it. The default value is false.

Parameters:

support: Takes boolean true or false.

Top | Methods | Enumerations

setXMLValidationMode

```
public void setXMLValidationMode(ENUMXMLValidationMode mode)
```

Sets the XML validation mode, which is an enumeration literal of ENUMXMLValidationMode.

Parameters:

mode: Is an enumeration literal of ENUMXMLValidationMode.

Top | Methods | Enumerations

setXQueryFileName

```
public void setXQueryFileName(String queryFile)
```

Sets the location, as a URL, of the XQuery file to be executed.

Parameters:

queryFile: The supplied string must be an absolute URL that gives the exact location of the XML file to use.

Top | Methods | Enumerations

setXQueryFromText

```
public void setXQueryFromText(String queryText)
```

Supplies the contents of the XQuery document as text.

Parameters:

queryText: The supplied string is the XQuery document to be processed.

Top | Methods | Enumerations

setXSDVersion

```
public void setXSDVersion(ENUMXSDVersion version)
```
Sets the XML Schema version against which the XML document will be validated.
Parameters:
`version`: Is an enumeration literal of `ENUMXSDVersion`.

Top | Methods | Enumerations

Enumerations
ENUMXQueryUpdatedXML
ENUMXQueryVersion

ENUMXQueryUpdatedXML

```
public enum ENUMXQueryUpdatedXML {
        eUpdatedDiscard
        eUpdatedWriteback
        eeUpdatedAsMainResult }
```

`ENUMXQueryVersion` takes one of the enumeration literals:
- `eUpdatedDiscard`: Updates are discarded and not written to file.
- `eUpdatedWriteback`: Updates are written to the input XML file specified with `setInputXMLFileName`.
- `eUpdatedAsMainResult`: Updates are written to the location specified by the `outputFile` parameter of `ExecuteUpdate`.

Used by (Interface::Method):
`XQuery` `setUpdatedXMLWriteMode`

Top | Methods | Enumerations

ENUMXQueryVersion

```
public enum ENUMXQueryVersion {
        eVersion10
        eVersion30 }
```

`ENUMXQueryVersion` takes one of the enumeration literals: `eVersion10`, `eVersion30`. These set the XQuery version to be used for processing (execution or validation).

Used by (Interface::Method):
`XQuery` `setVersion`

Top | Methods | Enumerations

6.2.5 XBRL

```
public interface XBRL
```

Description
Validates the supplied XBRL instance document or XBRL taxonomy document. The methods of the interface are described first, followed by its enumerations.

Utility class
A utility class for `FormulaParam` is defined. It holds two members and a constructor.

ParamValuePair

```
public class ParamValuePair
public class ParamValuePair
    {
        public String paramType;
        public String paramValue;
        public ParamValuePair( String type, String value )
        {
                        paramType = type;
                        paramValue = value;
        }
     };
```

Methods
The methods of the class are described below in alphabetical order. They are also organized into groups, according to functionality, for ease of reference.

Grouped by functionality

Processing

isValid(ENUM type)
isValid
getLastErrorMessage
setEvaluateReferencedParametersOnly
setParallelAssessment
setPythonScriptFile

Input Files

setInputFileName
setInputFileCollection
setInputFromText
setInputTextCollection

Formulas and Assertions
addAssertionForProcessing
addAssertionSetForProcessing
addFormulaArrayParameter
addFormulaForProcessing
addFormulaParameter(with NS)
addFormulaParameter
addFormulaParameterNamespace
clearFormulaParameterList
readFormulaAssertions
readFormulaOutput
setFormulaAssertionsAsXML
setFormulaAssertionsOutput
setFormulaOutput
evaluateFormula
setFormulaExtensionEnabled
setFormulaPreloadSchemas

Tables
addTableForProcessing
generateTables
setTableEliminateEmptyRows
setTableExtensionEnabled
setTableLinkbaseNamespace
setTableOutput
setTableOutputFormat
setTablePreloadSchemas

XML & XML Schema
setXincludeSupport
setSchemaImports
setSchemalocationHints
setSchemaMapping

General XBRL
setConceptLabelLinkrole
setConceptLabelRole
setGenericLabelLinkrole
setGenericLabelRole
setLabelLang
setDimensionExtensionEnabled
setPreloadSchemas
setTreatXBRLInconsistenciesAsErrors

addAssertionForProcessing

```
public void addAssertionForProcessing(String assertion)
```

Limits assertion execution to the given assertion only. Call multiple times to specify more than one assertion.

Parameters:

`assertion`: The supplied string holds the name of the assertion. Use `##none` for processing no assertion and `##all` for processing all assertions.

addAssertionSetForProcessing

```
public void addAssertionSetForProcessing(String assertionSet)
```

Limits assertion set execution to the given assertion set only. Call multiple times to specify more than one assertion set.

Parameters:

`assertionSet`: The supplied string holds the name of the assertion set. Use `##none` for processing no assertion set and `##all` for processing all assertion sets.

addFormulaArrayParameter

```
public void addFormulaArrayParameter(String type, String name, Object[]
values)
```

Adds an array-parameter used in the formula evaluation process.

Parameters:

`type`: A string that gives the default datatype of non-pair values inside array values. Default is `xs:string`.

`name`: A string that gives the parameter's name.

`values`: An array of values and datatype-value pairs.

For more information and code samples, see the section, XBRL Formula Parameters.

addFormulaForProcessing

```
public void addFormulaForProcessing(String formula)
```

Limits formula execution to the given formula only. Call multiple times to specify more than one formula.

Parameters:

`formula`: The supplied string holds the name of the formula. Use `##none` for processing no formula and `##all` for processing all formulas.

addFormulaParameter (with namespace) DEPRECATED

```
public void addFormulaParameter(String type, String name, String value, String
namespace)
```

Adds a parameter used in the formula evaluation process.

Parameters:

`type`: A string that gives the parameter's datatye.

`name`: A string that gives the parameter's name.

`value`: A string that gives the parameter's value.

`namespace`: A string that gives the parameter's namespace.

Note: If this method is used, the namespace is passed to `addFormulaParameterNamespace`.

addFormulaParameter

```
public void addFormulaParameter(String type, String name, String value)
```

Adds a parameter used in the formula evaluation process.

Parameters:

`type`: A string that gives the parameter's datatype.

`name`: A string that gives the parameter's name.

`value`: A string that gives the parameter's value.

addFormulaParameterNamespace

```
public void addFormulaParameter(String prefix, String URI)
```

Defines a namespace used in the QNames of parameter names, types, or values.

Parameters:

`prefix`: The namespace-prefix of values passed to `addFormulaArrayParameter`.

`URI`: The namespace URI.

addTableForProcessing

```
public void addTableForProcessing(String table)
```

Limits table generation to the given table only. Call multiple times to specify more than one table.

Parameters:

`table`: The supplied string holds the name of the table. Use `##none` for processing no table and `##all` for processing all tables.

clearFormulaParameterList

```
public void clearFormulaParameterList()
```

Clears the list of formula parameters created with the `addFormulaParameter` method.

evaluateFormula

```
public boolean evaluateFormula()
```

Returns the result of evaluating XBRL formulas in an XBRL instance file. The result is `true` on success, `false` on failure. If an error occurs, a `RaptorXMLException` is raised. Use the `getLastErrorMessage` method to access additional information.

Returns:

boolean `true` on success, `false` on failure.

generateTables

```
public boolean generateTables()
```

Evaluates XBRL tables in an instance file. The result is `true` on success, `false` on failure. If an error occurs, a `RaptorXMLException` is raised. Use the `getLastErrorMessage` method to access additional information.

Returns:

boolean `true` on success, `false` on failure.

getLastErrorMessage

```
public String getLastErrorMessage()
```

Retrieves the last error message from the XBRL engine.

Returns:

a string that is the last error message from the XBRL engine.

isValid

```
public boolean isValid(ENUMValidationType type)
```

Returns the result of validating the XBRL instance document or XBRL taxonomy document. The type of document to validate is specified by the `type` parameter, which takes an `ENUMValidationType` literal as its value. The result is `true` on success, `false` on failure. If an

error occurs, a RaptorXMLException is raised. Use the getLastErrorMessage method to access additional information.
Parameters:
type: An ENUMValidationType literal, which specifies whether the validation is of an XBRL instance document or of an XBRL taxonomy.
Returns:
boolean true on success, false on failure.

isValid
public boolean **isValid()**
Returns the result of validating the submitted XBRL document.The result is true on success, false on failure.
Returns:
boolean true on success, false on failure.

readFormulaAssertions
public String **readFormulaAssertions()**
Retrieves formula assertions from the specified file.
Returns:
a string containing the formula assertions.

readFormulaOutput
public String **readFormulaOutput()**
Evaluates formula assertions in the specified file and returns the result.
Returns:
a string that is an evaluation of the formula assertions.

setConceptLabelLinkrole
public void **setConceptLabelLinkrole(**String labelLinkrole**)**
Specifies the preferred extended link role to use when rendering concept labels.
Parameters:
labelLinkrole: The supplied string holds the preferred link role.

setConceptLabelRole
public void **setConceptLabelRole(**String labelRole**)**
Specifies the preferred label role to use when rendering concept labels.
Parameters:
labelRole: The supplied string holds the preferred label role. Default is: http://www.xbrl.org/2008/role/label.

setDimensionExtensionEnabled
public void **setDimensionExtensionEnabled(**boolean bEnable**)**
Enables XBRL Dimension extension validation. A value of true enables support; false disables it. Default is true.
Parameters:
bEnable: Takes boolean true or false.

setEvaluateReferencedParametersOnly

`public void setEvaluateReferencedParametersOnly(boolean bEnable)`

If `false`, forces evaluation of all parameters even if they are not referenced by any formulas/assertions/tables. Default is: `true`.

Parameters:

`bEnable`: Takes boolean `true` or `false`.

setFormulaAssertionsAsXML

`public void setFormulaAssertionsAsXML(boolean bEnable)`

Enables XML formatting of the assertion file when RaptorXML+XBRL is run with assertions enabled. A value of `true` enables XML output; `false` generates JSON output. Default is `false`.

Parameters:

`bEnable`: Takes boolean `true` or `false`.

setFormulaAssertionsOutput

`public void setFormulaAssertionsOutput(String outputFile)`

Sets the location of the file containing the retrieved formula assertions.

Parameters:

`outputFile`: The supplied string holds the full path of the output file.

setFormulaOutput

`public void setFormulaOutput(String outputFile)`

Sets the location of the file containing the output of formula evaluation.

Parameters:

`outputFile`: The supplied string holds the full path of the output file.

setFormulaExtensionEnabled

`public void setFormulaExtensionEnabled(boolean bEnable)`

Enables XBRL formula extensions for validation. A value of `true` enables support; `false` disables it. Default is `true`.

Parameters:

`bEnable`: Takes boolean `true` or `false`.

setFormulaPreloadSchemas

`public void setFormulaPreloadSchemas(boolean bEnable)`

Defines whether XBRL formula schemas will be preloaded. A value of `true` preloads the schemas; `false` does not. The default value is `false`.

Parameters:

`bEnable`: Takes boolean `true` or `false`.

setGenericLabelLinkrole

`public void setGenericLabelLinkrole(String labelLinkrole)`

Specifies the preferred extended link role to use when rendering generic labels.

Parameters:

`labelLinkrole`: The supplied string holds the preferred link role.

setGenericLabelRole

```
public void setGenericLabelRole(String labelRole)
```

Specifies the preferred label role to use when rendering generic labels.

Parameters:

`labelRole`: The supplied string holds the preferred label role. Default is: `http://www.xbrl.org/2008/role/label`.

setInputFileCollection

```
public void setInputFileCollection(Collection<?> fileCollection)
```

Supplies the collection of XBRL files that will be used as input data. The files are identified by their URLs.

Parameters:

`fileCollection`: A collection of strings, each of which is the absolute URL of an input XBRL file.

setInputFileName

```
public void setInputXMLFileName(String filePath)
```

Sets the location, as a URL, of the XBRL document to be validated.

Parameters:

`filePath`: The supplied string must be an absolute URL that gives the exact location of the XBRL file.

setInputFromText

```
public void setInputFromText(String inputText)
```

Supplies the contents of the XBRL document as text.

Parameters:

`inputText`: The supplied string is the content of the XBRL document to validate.

setInputTextCollection

```
public void setInputTextCollection(Collection<?> stringCollection)
```

Supplies the content of multiple XBRL files that will be used as input data.

Parameters:

`stringCollection`: A collection of strings, each of which is the content of an input XBRL document.

setLabelLang

```
public void setLabelLang(String labelLang)
```

Specifies the preferred label language to use when rendering labels.

Parameters:

`labelLang`: The supplied string holds the preferred label language. Default is: `en`.

setParallelAssessment

```
public void setParallelAssessment(boolean support)
```

Enables or disables the use of parallel assessment. A value of `true` enables parallel asessment; `false` disables it. The default value is `false`.
Parameters:
`support`: Takes boolean `true` or `false`.

setPreloadSchemas

```
public void setPreloadSchemas(boolean preload)
```

Defines whether XBRL 2.1 schemas will be pre-loaded. A value of `true` indicates preloads; `false` disables it. Default is `true`.
Parameters:
`support`: Takes boolean `true` or `false`. Default is `false`.

setPythonScriptFile

```
public void setPythonScriptFile(String file)
```

Sets the location, as a URL, of the Python script file.
Parameters:
`file`: The supplied string must be an absolute URL that gives the exact location of the Python file.

setSchemaImports

```
public void setSchemaImports(ENUMSchemaImports opt)
```

Specifies how schema imports are to be handled based on the attribute values of the `xs:import` elements. The kind of handling is specified by the `ENUMSchemaImports` literal that is selected.
Parameters:
`opt`: Holds the `ENUMSchemaImports` literal, which determines the handling of schema imports. See the description of `ENUMSchemaImports` for details.

setSchemalocationHints

```
public void setSchemalocationHints(ENUMLoadSchemalocation opt)
```

Specifies the mechanism to use to locate the schema. The mechanism is specified by the `ENUMLoadSchemalocation` literal that is selected.
Parameters:
`opt`: Holds the `ENUMLoadSchemalocation` literal, which determines which schema location mechanism to use. See the description of `ENUMLoadSchemalocation` for details.

setSchemaMapping

```
public void setSchemaMapping(ENUMSchemaMapping opt)
```

Sets what mapping to use in order to locate the schema. The mapping is specified by the `ENUMSchemaMapping` literal that is selected.
Parameters:
`opt`: Holds the `ENUMSchemaMapping` literal. See the description of `ENUMSchemaMapping` for details.

setTableEliminateEmptyRows

```
public void setTableEliminateEmptyRows(boolean bEnable)
```

Enables the elimination of empty table rows/columns in HTML output only. A value of `true` enables support; `false` disables it.
Parameters:
`bEnable`: Takes boolean `true` or `false`.

setTableExtensionEnabled

```
public void setTableExtensionEnabled(boolean bEnable)
```

Enables XBRL Table 1.0 extensions for validation. A value of `true` enables support; `false` disables it.
Parameters:
`bEnable`: Takes boolean `true` or `false`.

setTableLinkbaseNamespace

```
public void setTableLinkbaseNamespace(String namespace)
```

Enables the loading of table linkbases written with a previous draft specification. The `namespace` parameter specifies the table linkbase. Table linkbase validation, resolution, and layout is, however, always performed according to the Table Linkbase 1.0 Recommendation of 18 March 2014. Use `##detect` to enable auto-detection.
Parameters:
`namespace`: The following values are recognized:

```
##detect
http://xbrl.org/PWD/2013-05-17/table
http://xbrl.org/PWD/2013-08-28/table
http://xbrl.org/CR/2013-11-13/table
http://xbrl.org/PR/2013-12-18/table
http://xbrl.org/2014/table
```

setTableOutput

```
public void setTableOutput(String outputFile)
```

Sets the location of the file containing the output of table generation.
Parameters:
`outputFile`: The supplied string holds the full path of the output file.

setTableOutputFormat

```
public void setTableOutputAsXML(ENUMTableOutputFormat format)
```

Sets the format of the table output file. The format will be the value of `ENUMTableOutputFormat`.
Parameters:
`format`: Holds the value of `ENUMTableOutputFormat`.

setTablePreloadSchemas

```
public void setTablePreloadSchemas(boolean bEnable)
```

Enables preloading of schemas of the XBRL Table 1.0 specification. A value of `true` enables support; `false` disables it. Default is `false`.
Parameters:
`bEnable`: Takes boolean `true` or `false`.

setTreatXBRLInconsistenciesAsErrors

```
public void setTreatXBRLInconsistenciesAsErrors(boolean treat)
```

A value of `true` causes XBRL validation to fail if the file contains any inconsistencies as defined by the XBRL 2.1 specification. Default is `false`. When `false`, XBRL inconsistencies according to the XBRL 2.1 specification are not treated as errors.

Parameters:

`support`: Takes boolean `true` or `false`.

setXincludeSupport

```
public void setXIncludeSupport(boolean support)
```

Enables or disables the use of `XInclude` elements. A value of `true` enables XInclude support; `false` disables it. The default value is `false`.

Parameters:

`support`: Takes boolean `true` or `false`.

Top | Methods | Enumerations

Enumerations

ENUMValidationType

```
public enum ENUMValidationType {
        eValidateAny
        eValidateInstance
        eValidateTaxonomy }
```

`ENUMValidationType` contains the enumeration literal specifying what validation to carry out and, in the case of XML documents, whether validation is against a DTD or XSD.

- `eValidateAny`: The document type is detected automatically.
- `eValidateInstance`: Validates an XBRL instance document (`.xbrl` file extension).
- `eValidateTaxonomy`: Validates an XBRL taxonomy (`.xsd` file extension).

Used by (Interface::Method):

XBRL	isValid

ENUMTableOutputFormat

```
public enum ENUMTableOutputFormat {
        eFormatXML
        eFormatHTML }
```

`ENUMTableOutputFormat` contains the enumeration literal that specifies the output format of the document containing the generated tables.

- `eFormatXML`: The output document with the generated tables is in XML format.
- `eFormatHTML`: The output document with the generated tables is in HTML format.

Used by (Interface::Method):

XBRL	setTableOutputFormat

Top | Methods | Enumerations

6.2.6 RaptorXMLException

```
public interface RaptorXMLException
```

Description
Has a single method that generates the exception.

RaptorXMLException
```
public void RaptorXMLException(String message)
```
Generates an exception that contains information about an error that occurs during processing.
Parameters:
`message`: A string that provides information about the error.

Chapter 7

COM and .NET Interfaces

7 COM and .NET Interfaces

Two interfaces, one API
The COM and .NET interfaces of RaptorXML+XBRL Server use a single API: the COM/.NET API of RaptorXML+XBRL Server. The .NET interface is built as a wrapper around the COM interface.

You can use RaptorXML with:

- Scripting languages, such as JavaScript, via the COM interface
- Programming languages, such as C#, via the .NET interface

Organization of this section
This section is organized as follows:

- About the COM Interface, which describes how the COM interface works and steps you need to take to work with the COM interface
- About the .NET Interface, which describes how to set up your environment for working with the .NET interface.
- Programming Languages, which provides code listings in commonly used programming languages that show how to call RaptorXML functionality.
- The API Reference, which documents the object model, objects, and properties of the API.

7.1 About the COM Interface

RaptorXML+XBRL Server is automatically registered as a COM server object when RaptorXML +XBRL Server is installed. So it can be invoked from within applications and scripting languages that have programming support for COM calls. If you wish to change the location of the RaptorXML +XBRL Server installation package, it is best to de-install RaptorXML+XBRL Server and then re-install it at the required location. In this way the necessary de-registration and registration are carried out by the installer process.

Check the success of the registration

If the registration was successful, the Registry will contain the classes `RaptorXML.Server`. These two classes will typically be found under `HKEY_LOCAL_MACHINE\SOFTWARE\Classes`.

Code examples

A VBScript example showing how the RaptorXML API can be used via its COM interface is listed in the section Programming Languages. An example file corresponding to this listing is available in the `examples/API` folder of the RaptorXML application folder.

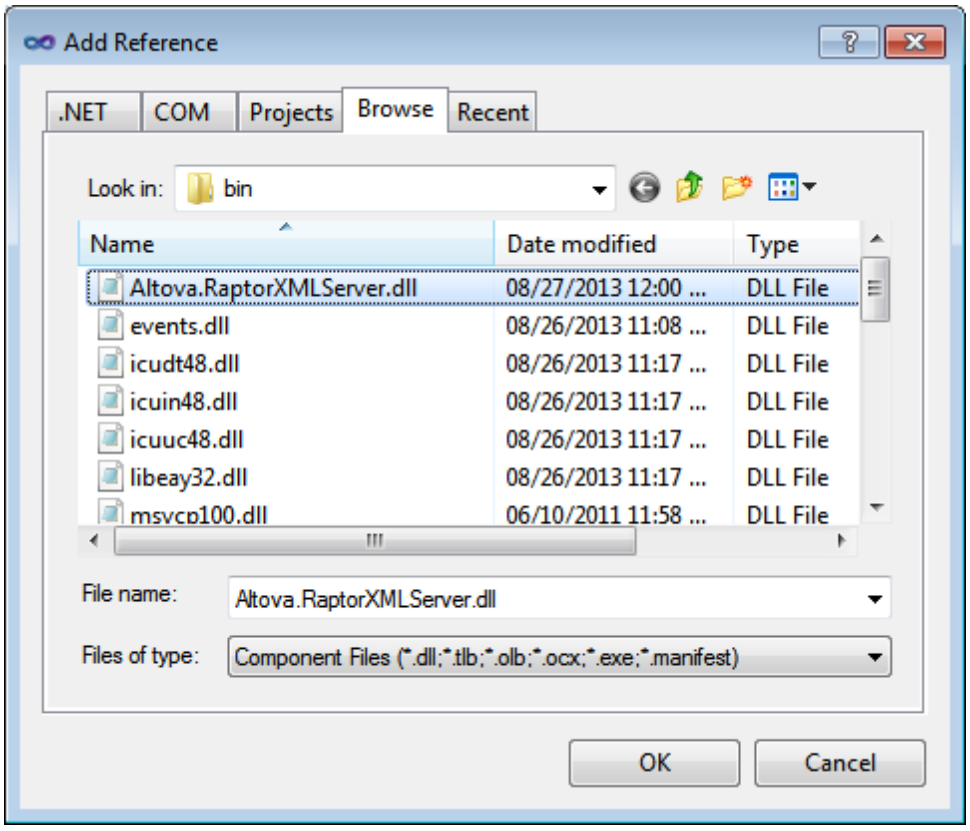
7.2 About the .NET Interface

The .NET interface is built as a wrapper around the RaptorXML COM interface. It is provided as a primary interop assembly signed by Altova; it uses the namespace `Altova.RaptorXMLServer`.

Adding the RaptorXML DLL as a reference to a Visual Studio .NET project
In order to use RaptorXML in your .NET project, add a reference to the RaptorXML DLL (`Altova.RaptorXMLServer.dll`) in your project. Your RaptorXML+XBRL Server installation contains a signed DLL file, named `Altova.RaptorXMLServer.dll`. This DLL file will automatically be added to the global assembly cache (GAC) when RaptorXML is installed using the RaptorXML installer. The GAC is typically in the folder: `C:\WINDOWS\assembly`.

To add the RaptorXML 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.



2. In the Browse tab, go to the folder: `<RaptorXML application folder>/bin`, select the RaptorXML DLL `Altova.RaptorXMLServer.dll`, and click **OK**.
3. Select the command View | Object Browser to see the objects of the RaptorXML API.

Once the `Altova.RaptorXMLServer.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: RaptorXML will automatically be registered as a COM server object during installation. There is no need for a manual registration.

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.

Code examples

A C# example and a Visual Basic .NET example showing how the RaptorXML API can be used via its .NET interface are listed in the section Programming Languages. The files corresponding to these listings are available in the `examples/API` folder of the RaptorXML application folder.

7.3 Programming Languages

Programming languages differ in the way they support COM and .NET access. A few examples for the most frequently used languages (*links below*) will help you get started. The code listings in this section show how basic functionality can be accessed. This basic functionality is included in the files in the `examples/API` folder of the RaptorXML+XBRL Server application folder.

VBScript
VBScript can be use to access the COM API of RaptorXML+XBRL Server. The VBScript listing demonstrates the following basic functionality:

- Connect to the RaptorXML+XBRL Server COM API
- Validate an XML file
- Carry out an XSL Transformation
- Do an XQuery execution

C#
C# can be used to access the .NET API of RaptorXML+XBRL Server. The C# code listing shows how to access the API for the following basic functionality:

- Connect to the RaptorXML+XBRL Server .NET API
- Validate an XML file
- Carry out an XSL Transformation
- Do an XQuery execution

Visual Basic .NET
Visual Basic.NET is different than C# in it syntax only, the .NET API accessing works in the same way. The Visual Basic code listing describes the following basic operations:

- Connect to the RaptorXML+XBRL Server .NET API
- Validate an XML file
- Carry out an XSL Transformation
- Do an XQuery execution

This section contains the following code examples:

For the COM interface

- An example in VBScript

For the .NET interface

- An example in C#
- An example in Visual Basic

7.3.1 COM Example: VBScript

The VBScript example below is structured into the following parts:

- Set up and initialize the RaptorXML COM object
- Validate an XML file
- Perform an XSLT transformation, return the result as a string
- Process an XQuery document, save the result in a file
- Set up the execution sequence of the script and its entry point

---

```
' The RaptorXML COM object
dim objRaptor

' Initialize the RaptorXML COM object
sub Init
	objRaptor = Null
	On Error Resume Next
	' Try to load the 32-bit COM object; do not throw exceptions if object is
not found
	Set objRaptor = WScript.GetObject( "", "RaptorXML.Server" )
	On Error Goto 0
	if ( IsNull( objRaptor ) ) then
		' Try to load the 64-bit object (exception will be thrown if not
found)
		Set objRaptor = WScript.GetObject( "", "RaptorXML_x64.Server" )
	end if
	' Configure the server: error reporting, HTTP server name and port (IPv6
localhost in this example)
	objRaptor.ErrorLimit = 1
	objRaptor.ReportOptionalWarnings = true
	objRaptor.ServerName = "::1"
	objRaptor.ServerPort = 8087
end sub
```

```
' Validate one file
sub ValidateXML
	' Get a validator instance from the Server object
	dim objXMLValidator
	Set objXMLValidator = objRaptor.GetXMLValidator()

	' Configure input data
	objXMLValidator.InputXMLFileName = "MyXMLFile.xml"

	' Validate; in case of invalid file report the problem returned by
RaptorXML
	if ( objXMLValidator.IsValid() ) then
		MsgBox( "Input string is valid" )
	else
		MsgBox( objXMLValidator.LastErrorMessage )
```

```
	end if
end sub
```

```
' Perform a transformation; return the result as a string
sub RunXSLT
	' Get an XSLT engine instance from the Server object
	dim objXSLT
	set objXSLT = objRaptor.GetXSLT

	' Configure input data
	objXSLT.InputXMLFileName = "MyXMLFile.xml"
	objXSLT.XSLFileName = "MyTransformation.xsl"

	' Run the transformation; in case of success the result will be returned,
in case of errors the engine returns an error listing
	MsgBox( objXSLT.ExecuteAndGetResultAsString() )
end sub
```

```
' Execute an XQuery; save the result in a file
sub RunXQuery
	' Get an XQuery engine instance from the Server object
	dim objXQ
	set objXQ = objRaptor.GetXQuery()

	' Configure input data
	objXQ.InputXMLFileName = "MyXMLFile.xml"
	objXQ.XQueryFileName = "MyQuery.xq"

	' Configure serialization (optional - for fine-tuning the result's
formatting)
	objXQ.OutputEncoding = "UTF8"
	objXQ.OutputIndent = true
	objXQ.OutputMethod = "xml"
	objXQ.OutputOmitXMLDeclaration = false

	' Run the query; the result will be serialized to the given path
	call objXQ.Execute( "MyQueryResult.xml" )
end sub
```

```
' Perform all sample functions
sub main
	Init
	ValidateXML
	RunXSLT
	RunXQuery
end sub

' Script entry point; run the main function
main
```


7.3.2 .NET Example: C#

The C# example below does the following:

- Set up and initialize the RaptorXML .NET object
- Validate an XML file
- Perform an XSLT transformation, return the result as a string
- Process an XQuery document, save the result in a file
- Set up the execution sequence of the code and its entry point

```
using System;
using System.Text;
using Altova.RaptorXMLServer;

namespace RaptorXMLRunner
{
    class Program
    {
    // The RaptorXML Server .NET object
       static ServerClass objRaptorXMLServer;

    // Initialize the RaptorXML Server .NET object
       static void Init()
       {
       // Allocate a RaptorXML Server object
          objRaptorXMLServer = new ServerClass();

       // Configure the server: error reporting, HTTP server name and port
       // (IPv6 localhost in this example)
          objRaptorXMLServer.ErrorLimit = 1;
          objRaptorXMLServer.ReportOptionalWarnings = true;
          objRaptorXMLServer.ServerName = "::1"
          objRaptorXMLServer.ServerPort = 8087
       }
```

```
       // Validate one file
          static void ValidateXML()
          {
          // Get a validator engine instance from the Server object
             XMLValidator objXMLValidator =
objRaptorXMLServer.GetXMLValidator();

          // Configure input data
             objXMLValidator.InputXMLFileName = "MyXMLFile.xml";

          // Validate; in case of invalid file,
             report the problem returned by RaptorXML
```

```
      if ( objXMLValidator.IsValid() )
         Console.WriteLine( "Input string is valid" );
      else
         Console.WriteLine( objXMLValidator.LastErrorMessage );
   }
```

```
// Perform an XSLT transformation, and
// return the result as a string
   static void RunXSLT()
   {
   // Get an XSLT engine instance from the Server object
      XSLT objXSLT = objRaptorXMLServer.GetXSLT();

      // Configure input data
         objXSLT.InputXMLFileName = "MyXMLFile.xml";
         objXSLT.XSLFileName = "MyTransformation.xsl";

     // Run the transformation.
     // In case of success, the result is returned.
     // In case of errors, an error listing
        Console.WriteLine( objXSLT.ExecuteAndGetResultAsString() );
   }
```

```
// Execute an XQuery, save the result in a file
   static void RunXQuery()
   {
   // Get an XQuery engine instance from the Server object
      XQuery objXQuery = objRaptorXMLServer.GetXQuery();

     // Configure input data
        objXQuery.InputXMLFileName = exampleFolder + "simple.xml";
        objXQuery.XQueryFileName = exampleFolder + "CopyInput.xq";

    // Configure serialization (optional, for better formatting)
       objXQuery.OutputEncoding = "UTF8"
       objXQuery.OutputIndent = true
       objXQuery.OutputMethod = "xml"
       objXQuery.OutputOmitXMLDeclaration = false

    // Run the query; result serialized to given path
       objXQuery.Execute( "MyQueryResult.xml" );
   }
```

```
   static void Main(string[] args)
```

```
            {
              try
              {
              // Entry point. Perform all functions
                 Init();
                 ValidateXML();
                 RunXSLT();
                 RunXQuery();
              }
              catch (System.Exception ex)
              {
                  Console.WriteLine( ex.Message );
                  Console.WriteLine( ex.ToString() );
              }
          }
      }
  }
```

7.3.3 .NET Example: Visual Basic .NET

The Visual Basic example below does the following:

- Set up and initialize the RaptorXML .NET object
- Validate an XML file
- Perform an XSLT transformation, return the result as a string
- Process an XQuery document, save the result in a file
- Set up the execution sequence of the code and its entry point

---

```
Option Explicit On
Imports Altova.RaptorXMLServer

Module RaptorXMLRunner

   ' The RaptorXML .NET object
     Dim objRaptor As Server

   ' Initialize the RaptorXML .NET object
     Sub Init()

       ' Allocate a RaptorXML object
         objRaptor = New Server()

       ' Configure the server: error reporting, HTTP server name and port (IPv6
localhost in this example)
         objRaptor.ErrorLimit = 1
         objRaptor.ReportOptionalWarnings = True
         objRaptor.ServerName = "::1"
         objRaptor.ServerPort = 8087
     End Sub
```

```
   ' Validate one file
     Sub ValidateXML()

       ' Get a validator instance from the RaptorXML object
         Dim objXMLValidator As XMLValidator
         objXMLValidator = objRaptor.GetXMLValidator()

       ' Configure input data
         objXMLValidator.InputXMLFileName = "MyXMLFile.xml"

       ' Validate; in case of invalid file report the problem returned by
RaptorXML
         If (objXMLValidator.IsValid()) Then
            Console.WriteLine("Input string is valid")
         Else
            Console.WriteLine(objXMLValidator.LastErrorMessage)
         End If
     End Sub
```

```
    ' Perform a transformation; return the result as a string
      Sub RunXSLT()

       ' Get an XSLT engine instance from the Server object
         Dim objXSLT As XSLT
         objXSLT = objRaptor.GetXSLT()

       ' Configure input data
         objXSLT.InputXMLFileName = "MyXMLFile.xml"
         objXSLT.XSLFileName = "MyTransformation.xsl"

       ' Run the transformation; in case of success the result will be returned,
in case of errors the engine returns an error listing
         Console.WriteLine(objXSLT.ExecuteAndGetResultAsString())
      End Sub
```

```
    ' Execute an XQuery; save the result in a file
      Sub RunXQuery()

       ' Get an XQuery engine instance from the Server object
         Dim objXQ As XQuery
         objXQ = objRaptor.GetXQuery()

       ' Configure input data
         objXQ.InputXMLFileName = "MyXMLFile.xml"
         objXQ.XQueryFileName = "MyQuery.xq"

       ' Configure serialization (optional - for fine-tuning the result's
formatting)
         objXQ.OutputEncoding = "UTF8"
         objXQ.OutputIndent = true
         objXQ.OutputMethod = "xml"
         objXQ.OutputOmitXMLDeclaration = false

       ' Run the query; the result will be serialized to the given path
         objXQ.Execute( "MyQueryResult.xml" )
      End Sub
```

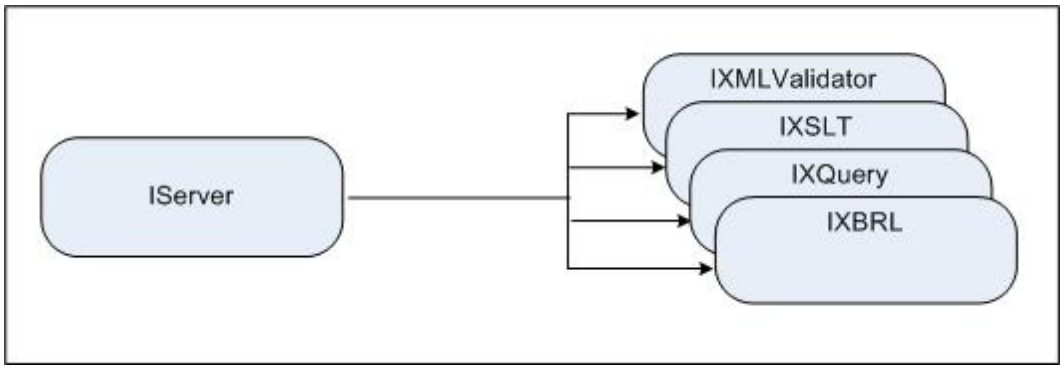
```
      Sub Main()
       ' Entry point; perform all sample functions
         Init()
         ValidateXML()
         RunXSLT()
         RunXQuery()
      End Sub

End Module
```


7.4 API Reference

This section describes the API specification: its object model and the details of its interfaces and enumerations.

The starting point for using the functionality of RaptorXML is the `IServer` interface. This object contains the objects that provide the RaptorXML functionality: XML validation, XBRL validation, XSLT transformations, and XQuery document processing. The object model of the RaptorXML API is depicted in the following diagram.



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

```
-- IServer
  |-- IXMLValidator
  |-- IXSLT
  |-- IXQuery
  |-- IXBRL
```

7.4.1 Interfaces

The following interfaces are defined. They are described in the sub-sections of this section.

IServer

IXMLValidator

IXSLT

IXQuery

IXBRL

IServer

The **IServer** interface provides methods to return interfaces of the respective RaptorXML engine: XML Validator, XBRL, XSLT and XQuery. The properties define the parameters of the interface.

Methods
IXMLValidator
IXBRL
IXSLT
IXQuery

Properties		
APIMajorVersion	GlobalResourcesFile	ServerName
APIMinorVersion	Is64Bit	ServerPath
APIServicePackVersion	MajorVersion	ServerPort
ErrorFormat	MinorVersion	ServicePackVersion
ErrorLimit	ProductName	UserCatalog
GlobalCatalog	ProductNameAndVersion	
GlobalResourceConfig	ReportOptionalWarnings	

Methods

The methods of the **IServer** interface return interfaces of the respective RaptorXML engine: XML Validator, XBRL, XSLT and XQuery.

IXMLValidator **GetXMLValidator()** [Top | Methods | Properties]

Returns an instance of the XML Validator Engine.

IXBRL **GetXBRL()** [Top | Methods | Properties]
Returns an instance of the XBRL Engine.

IXSLT **GetXSLT()** [Top | Methods | Properties]
Returns an instance of the XSLT Engine.

IXQuery **GetXQuery()** [Top | Methods | Properties]
Returns an instance of the XQuery Engine.

Properties
The properties of the **IServer** interface are described below in alphabetical order. The table arranges the properties in groups for ease of reference. Note that string inputs to be interpreted as URLs must provide absolute paths. If a relative path is used, a mechanism to resolve the relative path should be defined in the calling module.

Errors and Warnings	Catalogs	Global Resources	HTTP Server
ErrorFormat	GlobalCatalog	GlobalResourceConfig	ServerName
ErrorLimit	UserCatalog	GlobalResourcesFile	ServerPath
ReportOptionalWarnings			ServerPort

Product Information	
ProductName	Is64Bit
ProductNameAndVersion	APIMajorVersion
MajorVersion	APIMinorVersion
MinorVersion	APIServicePackVersion
ServicePackVersion	

int **APIMajorVersion** [Top | Methods | Properties]
Returns the major version of the API as an integer. The major version of the API could be different from the product's major version if the API is connected to another server.

---

int **APIMinorVersion** [Top | Methods | Properties]
Returns the minor version of the API as an integer. The minor version of the API could be different

from the product's minor version if the API is connected to another server.

`int` **APIServicePackVersion** [Top | Methods | Properties]
Returns the service pack version of the API as an integer. The service pack version of the API could be different from the product's service pack version if the API is connected to another server.

ENUMErrorFormat **ErrorFormat** [Top | Methods | Properties]
Sets the RaptorXML error format and is an ENUMErrorFormat literal (`Text | ShortXML | LongXML`).

`int` **ErrorLimit** [Top | Methods | Properties]
Configures the RaptorXML validation error limit. Type is `uint`. If the error limit is reached, execution is halted. The default value is `100`.

`string` **GlobalCatalog** [Top | Methods | Properties]
Specifies the location of the main (entry-point) catalog file. The supplied string must be an absolute URL that gives the exact location of the catalog file to use.

`string` **GlobalResourceConfig** [Top | Methods | Properties]
Specifies the active configuration of the global resource to be used.

`string` **GlobalResourcesFile** [Top | Methods | Properties]
Specifies the global resource file. The supplied string must be an absolute URL that gives the exact location of the global resources file to use.

`bool` **Is64Bit** [Top | Methods | Properties]
Checks if the application is a 64-bit executable. *Example:* For `Altova RaptorXML+XBRL Server 2015r2sp1(x64)`, returns `true`.

`int` **MajorVersion** [Top | Methods | Properties]
Returns the major version of the product as an integer. *Example:* For `Altova RaptorXML+XBRL Server 2014r2sp1(x64)`, returns `16` (the difference between the major version (`2014`) and the initial year `1998`).

`int` **MinorVersion** [Top | Methods | Properties]
Returns the minor version of the product as an integer. *Example:* For `Altova RaptorXML+XBRL`

Server 2015r2sp1(x64), returns 2 (from the minor version number r2).

---

string **ProductName** [Top | Methods | Properties]
Returns the name of the product as a string. *Example:* For Altova RaptorXML+XBRL Server 2015r2sp1(x64), returns Altova RaptorXML+XBRL Server.

---

string **ProductNameAndVersion** [Top | Methods | Properties]
Returns the name and version of the product as a string. *Example:* For Altova RaptorXML+XBRL Server 2015r2sp1(x64), returns Altova RaptorXML+XBRL Server 2015r2sp1(x64).

---

bool **ReportOptionalWarnings** [Top | Methods | Properties]
Enables or disables the reporting of warnings. A value of true enables warnings; false disables them.

---

string **ServerName** [Top | Methods | Properties]
Sets the name of the HTTP server. A RaptorXMLException is raised if an error occurs.

---

string **ServerPath** [Top | Methods | Properties]
Specifies, in the form of a URL, the path to the HTTP server. A RaptorXMLException is raised if an error occurs.

---

int **ServerPort** [Top | Methods | Properties]
Specifies the server port of the HTTP server. Type is ushort. A RaptorXMLException is raised if an error occurs.

---

int **ServicePackVersion** [Top | Methods | Properties]
Returns the service pack version of the product as an integer. *Example:* For RaptorXML+XBRL Server 2015r2sp1(x64), returns 1 (from the service pack version number sp1).

---

string **UserCatalog** [Top | Methods | Properties]
Specifies, as a URL, the location of the user-defined catalog file. The supplied string must be an absolute URL that gives the exact location of the user catalog file to use.

IXMLValidator

The **IXMLValidator** interface provides methods to test:

- The validity of an XML document, DTD, or XML Schema document: IsValid. XML documents can be validated against a DTD or XML Schema, references to which can be

within the XML document or be supplied via the code.
- The well-formedness of an XML document: IsWellFormed.

Both methods return boolean TRUE or FALSE. The properties define the parameters of the interface.

Methods
IsValid
IsWellFormed

Properties		
AssessmentMode	InputXMLFromText	SchemalocationHints
DTDFileName	LastErrorMessage	SchemaMapping
DTDFromText	PythonScriptFile	SchemaTextArray
EnableNamespaces	SchemaFileArray	Streaming
InputFileArray	SchemaFileName	XincludeSupport
InputTextArray	SchemaFromText	XMLValidationMode
InputXMLFileName	SchemaImports	XSDVersion

Methods

The two methods of the **IXMLValidator** interface are **IsValid** and **IsWellFormed**. They test, respectively, the validity and well-formedness of the specified document. Both methods return boolean true or false.

---

bool **IsValid**(ENUMValidationType nType) [Top | Methods | Properties]

- Returns the result of the validation specified by the value of ENUMValidationType. Returns true on success, false on failure.
- nType is the value of ENUMValidationType. The validation type specifies whether XML is to be validated against a DTD or XSD, or whether a DTD or XSD is to be validated. Default is eValidateAny, which indicates that the type of document shoud be determined by RaptorXML automatically.
- If an error occurs during execution, a RaptorXMLException is raised. Use the LastErrorMessage operation to access additional information.

---

bool **IsWellFormed**(ENUMWellformedCheckType nType) [Top | Methods | Properties]

- Returns the result of the well-formedness check specified by the value of ENUMWellformedCheckType. Returns true on success, false on failure.
- nType is the value of ENUMWellformedCheckType. Its value specifies whether an XML document or DTD document is to be checked. Default is eWellformedAny.
- If an error occurs during execution, a RaptorXMLException is raised. Use the LastErrorMessage operation to access additional information.

Properties

The properties of the **IXMLValidator** interface are described below in alphabetical order. The table arranges the properties in groups for ease of reference. Note that string inputs to be interpreted as URLs must provide absolute paths. If a relative path is used, a mechanism to resolve the relative path should be defined in the calling module.

Data Files	Schema Files	Processing
InputFileArray	DTDFileName	AssessmentMode
InputTextArray	DTDFromText	EnableNamespaces
InputXMLFileName	SchemaFileArray	LastErrorMessage
InputXMLFromText	SchemaFileName	PythonScriptFile
	SchemaFromText	Streaming
	SchemaImports	XincludeSupport
	SchemalocationHints	XMLValidationMode
	SchemaMapping	XSDVersion
	SchemaTextArray	

ENUMAssessmentMode **AssessmentMode** [Top | Methods | Properties]
Sets the assessment mode of the XML validator (strict or lax), as specified by ENUMAssessmentMode literals.

---

string **DTDFileName** [Top | Methods | Properties]
Specifies the external DTD document to use for validation. The supplied string must be an absolute URL that gives the base location of the DTD to use.

---

string **DTDFromText** [Top | Methods | Properties]
Provides the entire DTD as a string.

bool **EnableNamespaces** [Top | Methods | Properties]
Enables namespace-aware processing. This is useful for checking the XML instance for errors due to incorrect namespaces. A value of `true` enables namespace-aware processing; `false` disables it. Default value is `false`.

object **InputFileArray** [Top | Methods | Properties]
Provides an array of the URLs of the XML files to be used as input data. The property supplies an object containing, as strings, the absolute URLs of each of the XML files.

object **InputTextArray** [Top | Methods | Properties]
Provides an array of the URLs of the text-files to be used as input data. The property supplies an object containing, as strings, the absolute URLs of each of the text files.

string **InputXMLFileName** [Top | Methods | Properties]
Specifies the XML file to be validated. The supplied string must be an absolute URL that gives the base location of the XML file to use.

string **InputXMLFromText** [Top | Methods | Properties]
Supplies, as a text string, the contents of the XML document to be validated.

string **LastErrorMessage** [Top | Methods | Properties]
Retrieves the last error message from the RaptorXML Engine as a string.

bool **ParallelAssessment** [Top | Methods | Properties]
Enables/disables parallel schema validity assessment.

string **PythonScriptFile** [Top | Methods | Properties]
Specifies the Python script file that provides additional processing of the XML or XSD file submitted for validation. The supplied string must be an absolute URL that gives the base location of the Python script.

object **SchemaFileArray** [Top | Methods | Properties]
Provides an array of the URLs of the XSD files to be used as external XML Schemas. The property supplies an object containing, as strings, the absolute URLs of each of the XML Schema files.

string **SchemaFileName** [Top | Methods | Properties]

Specifies the external XML Schema file to be used for validation. The supplied string must be an absolute URL that gives the base location of the XML Schema file to use.

---

string **SchemaFromText** [Top | Methods | Properties]
Supplies, as a text string, the contents of the XML Schema document to be used for validation.

ENUMSchemaImports **SchemaImports** [Top | Methods | Properties]
Specifies how schema imports are to be handled according to the attribute values of the xs:import elements. The handling is specified by the ENUMSchemaImports literal that is selected.

ENUMLoadSchemalocation **SchemalocationHints** [Top | Methods | Properties]
Specifies the mechanism to use to locate the schema. The mechanism is specified by the ENUMLoadSchemalocation literal that is selected.

ENUMSchemaMapping **SchemaMapping** [Top | Methods | Properties]
Sets what mapping to use in order to locate the schema. The mapping is specified by the ENUMSchemaMapping literal that is selected.

---

object **SchemaTextArray** [Top | Methods | Properties]
Provides an array of strings that are the XSD files to be used as external XML Schemas. The property supplies an object containing, as strings, the text strings of each of the XML Schema files.

---

bool **Streaming** [Top | Methods | Properties]
Enables streaming validation. In streaming mode, data stored in memory is minimized and processing is faster. A value of true enables streaming validation; false disables it. Default is true.

---

bool **XincludeSupport** [Top | Methods | Properties]
Enables the use of XInclude elements. A value of true enables XInclude support; false disables it. The default value is false.

ENUMXMLValidationMode **XMLValidationMode** [Top | Methods | Properties]
Sets the XML validation mode (validation or well-formed check). The mode is that specified by the ENUMXMLValidationMode literal.

ENUMXSDVersion **XSDVersion** [Top | Methods | Properties]

Specifies the XML Schema version against which the XML document will be validated. Values are the ENUMXSDVersion literals.

IXSLT

The **IXSLT** interface provides methods and properties to execute an XSLT 1.0, XSLT 2.0, or XSLT 3.0 transformation. Results can be saved to a file or returned as a string. The interface also enables XSLT parameters to be passed to the XSLT stylesheet. The URLs of XML and XSLT files can be supplied as strings via the properties of the interface. 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.

Note: The XSLT 2.0 or 3.0 Engine of RaptorXML can be used in its backward compatibility mode to process an XSLT 1.0 stylesheet. The output, however, could be different than that produced by the XSLT 1.0 Engine processing the same XSLT 1.0 stylesheet.

Methods
IsValid
Execute
ExecuteAndGetResultAsString
ExecuteAndGetResultAsStringWithBaseOutputURI
AddExternalParameter
ClearExternalParameterList

Properties		
ChartExtensionsEnabled	JavaBarcodeExtensionLocation	SchemaMapping
DotNetExtensionsEnabled	JavaExtensionsEnabled	StreamingSerialization
EngineVersion	LastErrorMessage	XincludeSupport
IndentCharacters	LoadXMLWithPSVI	XMLValidationMode
InitialTemplateMode	NamedTemplateEntryPoint	XSDVersion
InputXMLFileName	SchemaImports	XSLFileName
InputXMLFromText	SchemalocationHints	XSLFromText

Methods

The methods of the **IXSLT** interface are described below. Note that string inputs to be interpreted

as URLs must provide absolute paths. If a relative path is used, a mechanism to resolve the relative path should be defined in the calling module.

Methods
IsValid
Execute
ExecuteAndGetResultAsString
ExecuteAndGetResultAsStringWithBaseOutputURI
AddExternalParameter
ClearExternalParameterList

bool **IsValid()** [Top | Methods | Properties]

- Returns the result of validating the XSLT stylesheet according to the XSLT specification named in ENUMXSLTVersion (see the EngineVersion property). The result is true on success, false on failure.
- If an error occurs, a RaptorXMLException is raised. Use the LastErrorMessage operation to access additional information.

---

bool **Execute(string bstrResultFileName)** [Top | Methods | Properties]

- Executes the XSLT transformation according to the XSLT specification named in ENUMXSLTVersion (see the EngineVersion property), and saves the result to an output file.
- The output file is defined by bstrResultFileName, which is a string that provides the URL of the output file.
- The result is true on success, false on failure.
- If an error occurs during the transformation, a RaptorXMLException is raised. Use the LastErrorMessage operation to access additional information.

---

string **ExecuteAndGetResultAsString()** [Top | Methods | Properties]

- Executes the XSLT transformation according to the XSLT specification named in ENUMXSLTVersion (see the EngineVersion property), and returns the transformation result as a string.
- If an error occurs during the transformation, a RaptorXMLException is raised. Use the LastErrorMessage operation to access additional information.

`string` **`ExecuteAndGetResultAsStringWithBaseOutputURI(`**`string bstrBaseURI`**`)` [Top | Methods | Properties]

- Executes the XSLT transformation according to the XSLT specification named in ENUMXSLTVersion (see the EngineVersion property), and returns the transformation result as a string at the location defined by the base URI (the string `bstrBaseURI`).
- If an error occurs during the transformation, a `RaptorXMLException` is raised. Use the LastErrorMessage operation to access additional information.

`void` **`AddExternalParameter(`**`string bstrName, string bstrValue`**`)` [Top | Methods | Properties]

- Adds the name and value of an external parameter: `bstrName` and `bstrValue` are strings.
- Each external parameter and its value must be specified in a separate call to the method. Parameters must be declared in the XSLT document, optionally with a type declaration. Whatever the type declaration in the XSLT document, no special delimiter is needed when the parameter value is submitted with `AddExternalParameter`.

`void` **`ClearExternalParameterList()`** [Top | Methods | Properties]

- Clears the external parameters list created with the AddExternalParameter method.

Properties

The properties of the **`IXSLT`** interface are described below in alphabetical order. The table arranges the properties in groups for ease of reference. Note that string inputs to be interpreted as URLs must provide absolute paths. If a relative path is used, a mechanism to resolve the relative path should be defined in the calling module.

XML	XSLT	Schema
InputXMLFileName	EngineVersion	SchemaImports
InputXMLFromText	XSLFileName	SchemalocationHints
LoadXMLWithPSVI	XSLFromText	SchemaMapping
XincludeSupport		XSDVersion
XMLValidationMode		
Processing	**Extensions**	
IndentCharacters	ChartExtensionsEnabled	

InitialTemplateMode	DotNetExtensionsEnabled
LastErrorMessage	JavaBarcodeExtensionLocation
NamedTemplateEntryPoint	JavaExtensionsEnabled
StreamingSerialization	

`bool` **ChartExtensionsEnabled** [Top | Methods | Properties]
Enables or disables Altova's chart extension functions. A value of `true` enables chart extensions; `false` disables them. Default value is `true`.

`bool` **DotNetExtensionsEnabled** [Top | Methods | Properties]
Enables or disables Visual Studio .NET extension functionss. A value of `true` enables .NET extensions; `false` disables them. Default value is `true`.

ENUMXSLTVersion **EngineVersion** [Top | Methods | Properties]
Specifies the XSLT version to use (1.0, 2.0, or 3.0). The property value is an ENUMXSLTVersion literal.

`string` **IndentCharacters** [Top | Methods | Properties]
Sets the character string that will be used as indentation.

`string` **InitialTemplateMode** [Top | Methods | Properties]
Sets the initial mode for XSLT processing. Templates with a mode value equal to the submitted string will be processed.

`string` **InputXMLFileName** [Top | Methods | Properties]
Specifies the location of the XML file to be transformed. The supplied string must be an absolute URL that gives the exact location of the XML file to use.

`string` **InputXMLFromText** [Top | Methods | Properties]
Supplies, as a text string, the contents of the XML document to be transformed.

`string` **JavaBarcodeExtensionLocation** [Top | Methods | Properties]
Specifies the location of the barcode extension file. See the section on Altova's barcode extension functions for more information. The supplied string must be an absolute URL that gives the base location of the file to use.

---

bool **JavaExtensionsEnabled** [Top | Methods | Properties]
Enables or disables Java extensions. A value of true enables Java extensions; false disables them. Default value is true.

---

string **LastErrorMessage** [Top | Methods | Properties]
Retrieves the last error message from the RaptorXML Engine as a string.

---

bool **LoadXMLWithPSVI** [Top | Methods | Properties]
Enables the option to load and use the Post Schema Validation Infoset (PSVI). If the PSVI is loaded, information obtained from the schema can be used to qualify data in the XML document. A value of true enables PSVI loading; false disables it.

---

string **NamedTemplateEntryPoint** [Top | Methods | Properties]
Specifies the name, as a string, of the named template to use as an entry point for the transformation.

ENUMSchemaImports **SchemaImports** [Top | Methods | Properties]
Specifies how schema imports are to be handled according to the attribute values of the xs:import elements. The handling is specified by the ENUMSchemaImports literal that is selected.

ENUMLoadSchemalocation **SchemalocationHints** [Top | Methods | Properties]
Specifies the mechanism to use to locate the schema. The mechanism is specified by the ENUMLoadSchemalocation literal that is selected.

ENUMSchemaMapping **SchemaMapping** [Top | Methods | Properties]
Sets what mapping to use in order to locate the schema. The mapping is specified by the ENUMSchemaMapping literal that is selected.

---

bool **StreamingSerialization** [Top | Methods | Properties]
Enables streaming serialization. In streaming mode, data stored in memory is minimized and processing is faster. A value of true enables streaming serialization; false disables it.

---

bool **XincludeSupport** [Top | Methods | Properties]
Enables the use of XInclude elements. A value of true enables XInclude support; false disables it. The default value is false.

ENUMXMLValidationMode **XMLValidationMode** [Top | Methods | Properties]

Sets the XML validation mode (validation or well-formed check). The mode is that specified by the ENUMXMLValidationMode literal.

ENUMXSDVersion **XSDVersion** [Top | Methods | Properties]
Specifies the XML Schema version against which the XML document will be validated. Values are the ENUMXSDVersion literals.

---

string **XSLFileName** [Top | Methods | Properties]
Specifies the XSLT file to be used for the transformation. The supplied string must be an absolute URL that gives the location of the XSLT file to use.

---

string **XSLFromText** [Top | Methods | Properties]
Supplies, as a text string, the contents of the XSLT document to be used for the transformation.

IXQuery

The **IXQuery** interface provides methods and properties to execute an XQuery 1.0 or XQuery 3.0 document. Results can be saved to a file or returned as a string. The interface also enables external XQuery variables to be passed to the XQuery document. The URLs of XQuery and XML files can be supplied as strings via the properties of the interface. Alternatively, the XML and XQuery documents can be constructed within the code as text strings.

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

Methods
IsValid
IsValidUpdate
Execute
ExecuteUpdate
ExecuteAndGetResultAsString
ExecuteUpdateAndGetResultAsString
AddExternalVariable
ClearExternalParameterList

Properties			
ChartExtensionsEnabled	InputXMLFromText	OutputEncoding	XMLValidationMode
DotNetExtensionsEnabled	JavaBarcodeExtensionLocation	OutputIndent	XQueryFileName

EngineVersion	JavaExtensionsEnabled	OutputMethod	XQueryFromText
IndentCharacters	LastErrorMessage	OutputOmitXMLDeclaration	XSDVersion
InputXMLFileName	LoadXMLWithPSVI	XincludeSupport	

Methods

The methods of the **IXQuery** interface are described below. Note that string inputs to be interpreted as URLs must provide absolute paths. If a relative path is used, a mechanism to resolve the relative path should be defined in the calling module.

Methods
IsValid
IsValidUpdate
Execute
ExecuteUpdate
ExecuteAndGetResultAsString
ExecuteUpdateAndGetResultAsString
AddExternalVariable
ClearExternalParameterList

`bool` **IsValid()** [Top | Methods | Properties]

- Returns the result of validating the XQuery document according to the XQuery specification named in ENUMXQueryVersion (see the EngineVersion property). The result is `true` on success, `false` on failure.
- If an error occurs, a `RaptorXMLException` is raised. Use the LastErrorMessage operation to access additional information.

`bool` **IsValidUpdate()** [Top | Methods | Properties]

- Returns the result of validating the XQuery Update document according to the XQuery specification named in ENUMXQueryVersion (see the EngineVersion property). The result is `true` on success, `false` on failure.
- If an error occurs, a `RaptorXMLException` is raised. Use the LastErrorMessage operation to access additional information.

bool **Execute(string bstrOutputFile)** [Top | Methods | Properties]

- Executes the XQuery according to the XQuery specification named in ENUMXQueryVersion (see the EngineVersion property), and saves the result to an output file.
- The output file is defined by bstrOutputFile, which is a string that provides the URL of the output file.
- Boolean true is returned on success, false on failure.
- If an error occurs during the transformation, a RaptorXMLException is raised. Use the LastErrorMessage operation to access additional information.

bool **ExecuteUpdate(string bstrOutputFile)** [Top | Methods | Properties]

- Executes the XQuery update according to the XQuery Update specification named in ENUMXQueryVersion (see the EngineVersion property), and saves the result to an output file.
- The output file is defined by bstrOutputFile, which is a string that provides the URL of the output file.
- Boolean true is returned on success, false on failure.
- If an error occurs during the transformation, a RaptorXMLException is raised. Use the LastErrorMessage operation to access additional information.

string **ExecuteAndGetResultAsString()** [Top | Methods | Properties]

- Executes the XQuery transformation according to the XQuery specification named in ENUMXQueryVersion (see the EngineVersion property), and returns the transformation result as a string.
- If an error occurs during the transformation, a RaptorXMLException is raised. Use the LastErrorMessage operation to access additional information.

string **ExecuteUpdateAndGetResultAsString()** [Top | Methods | Properties]

- Executes the XQuery update according to the XQuery Update specification named in ENUMXQueryVersion (see the EngineVersion property), and returns the transformation result as a string.
- If an error occurs during the transformation, a RaptorXMLException is raised. Use the LastErrorMessage operation to access additional information.

`void` **`AddExternalVariable(`**`string bstrName, string bstrValue`**`)` [Top | Methods | Properties]

- Adds the name and value of an external variable: `bstrName` and `bstrValue` are strings.
- Each external variable and its value must be specified in a separate call to the method. Variables must be declared in the XQuery document, optionally with a type declaration. If the variable value is a string, enclose the value in single quotes.

`void` **`ClearExternalVariableList()`** [Top | Methods | Properties]

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

Properties

The properties of the **`IXQuery`** interface are described below in alphabetical order. The table arranges the properties in groups for ease of reference. Note that string inputs to be interpreted as URLs must provide absolute paths. If a relative path is used, a mechanism to resolve the relative path should be defined in the calling module.

XML	XQuery	Processing	Extensions
InputXMLFileName	EngineVersion	IndentCharacters	ChartExtensionsEnabled
KeepFormatting	XQueryFileName	LastErrorMessage	DotNetExtensionsEnabled
InputXMLFromText	XQueryFromText	OutputEncoding	JavaBarcodeExtensionLocation
LoadXMLWithPSVI		OutputIndent	JavaExtensionsEnabled
XincludeSupport		OutputMethod	
XMLValidationMode		OutputOmitXMLDeclaration	
XSDVersion		UpdatedXMLWriteMode	

`bool` **`ChartExtensionsEnabled`** [Top | Methods | Properties]

Enables or disables Altova's chart extension functions. A value of `true` enables chart extensions; `false` disables them. Default value is `true`.

`bool` **`DotNetExtensionsEnabled`** [Top | Methods | Properties]

Enables or disables Visual Studio .NET extension functionss. A value of `true` enables .NET

extensions; `false` disables them. Default value is `true`.

ENUMXQueryVersion **EngineVersion** [Top | Methods | Properties]
Specifies the XQuery version to use (1.0 or 3.0). The property value is an ENUMXQueryVersion literal.

`string` **IndentCharacters** [Top | Methods | Properties]
Sets the character string that will be used as indentation.

`string` **InputXMLFileName** [Top | Methods | Properties]
Specifies the location of the XML file to be processed. The supplied string must be an absolute URL that gives the exact location of the XML file to use.

`string` **InputXMLFromText** [Top | Methods | Properties]
Supplies, as a text string, the contents of the XML document to be processed.

`string` **JavaBarcodeExtensionLocation** [Top | Methods | Properties]
Specifies the location of the barcode extension file. See the section on Altova's barcode extension functions for more information. The supplied string must be an absolute URL that gives the base location of the file to use.

`bool` **JavaExtensionsEnabled** [Top | Methods | Properties]
Enables or disables Java extensions. A value of `true` enables Java extensions; `false` disables them. Default value is `true`.

`bool` **KeepFormatting** [Top | Methods | Properties]
Specifies whether the formatting of the original document should be kept (as far as possible) or not. A value of `true` keeps formatting; `false` does not keep formatting. Default value is `true`.

`string` **LastErrorMessage** [Top | Methods | Properties]
Retrieves the last error message from the RaptorXML Engine as a string.

`bool` **LoadXMLWithPSVI** [Top | Methods | Properties]
Enables or disables the option to load and use the Post Schema Validation Infoset (PSVI). If the PSVI is loaded, information obtained from the schema can be used to qualify data in the XML document. A value of `true` enables PSVI loading; `false` disables it.

---

string **OutputEncoding** [Top | Methods | Properties]
Sets the encoding for the result document. Use an official IANA encoding name, such as UTF-8, UTF-16, US-ASCII, ISO-8859-1, as a string.

---

bool **OutputIndent** [Top | Methods | Properties]
Enables or disables indentation in the output document. A value of true enables indentation; false disables it.

---

string **OutputMethod** [Top | Methods | Properties]
Specifies the serialization of the output document. Valid values are: xml | xhtml | html | text. Default value is xml.

---

bool **OutputOmitXMLDeclaration** [Top | Methods | Properties]
Enables/disables the inclusion of the XML declaration in the result document. A value of true omits the declaration; false includes it. Default value is false.

ENUMXQueryUpdatedXML **UpdatedXMLWriteMode** [Top | Methods | Properties]
Specifies how updates to the XML file are handled. The property value is an ENUMXQueryUpdatedXML literal.

---

bool **XincludeSupport** [Top | Methods | Properties]
Enables or disables the use of XInclude elements. A value of true enables XInclude support; false disables it. The default value is false.

ENUMXMLValidationMode **XMLValidationMode** [Top | Methods | Properties]
Sets the XML validation mode (validation or well-formed check). The mode is that specified by the ENUMXMLValidationMode literal.

---

string **XQueryFileName** [Top | Methods | Properties]
Specifies the XQuery file to use. The supplied string must be an absolute URL that gives the location of the XSLT file to use.

---

string **XQueryFromText** [Top | Methods | Properties]
Supplies, as a text string, the contents of the XQuery document to use.

ENUMXSDVersion **XSDVersion** [Top | Methods | Properties]

Specifies the XML Schema version against which the XML document will be validated. Values are the ENUMXSDVersion literals.

IXBRL

The **IXBRL** interface provides methods to validate XBRL instance and taxonomy documents, as well as formulas. Results are boolean `true` or `false`. The interface also enables formula parameters to be passed through for formula evaluation. Formula assertions and output can also read, and returned as strings. The properties define the parameters of the interface.

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.

Structures

The following structure is defined.

```
public struct XBRLParamValuePair
{
    String ParamType;
    String ParamValue;
};
```

Methods

The methods of the **IXBRL** interface are described below. Note that string inputs to be interpreted as URLs must provide absolute paths. If a relative path is used, a mechanism to resolve the relative path should be defined in the calling module.

`bool` **IsValid(**ENUMXBRLValidationType `nType`**)** [Top | Methods | Properties]

- Returns the result of validating the XBRL instance document or XBRL taxonomy document.
- `nType` is the value of ENUMXBRLValidationType. The validation type specifies whether the XBRL instance document or XBRL taxonomy is to be validated. Default is `eValidateXBRLAny`, which indicates that the type of document shoud be determined by RaptorXML automatically.
- If an error occurs during execution, a `RaptorXMLException` is raised. Use the LastErrorMessage operation to access additional information.

`bool` **`EvaluateFormula()`** [Top | Methods | Properties]

- Evaluates XBRL formulas in an XBRL instance document. Returns `true` if valid, `false` if any formula is invalid.
- If an error occurs during execution, a `RaptorXMLException` is raised. Use the `LastErrorMessage` operation to access additional information.

`bool` **`GenerateTables()`** [Top | Methods | Properties]

- Evaluates XBRL table in an XBRL instance document. Returns `true` on success, `false` on failure.
- If an error occurs during execution, a `RaptorXMLException` is raised. Use the `LastErrorMessage` operation to access additional information.

`void` **`AddFormulaArrayParameter(`**`string sDefaultType, string sName, object[] variantValues`**`)` [Top | Methods | Properties]

- Adds an array-parameter used in the formula evaluation process..
- All the arguments are strings: `sDefaultType` is the default datatype of non-pair values inside array values. Default is `xs:string`; `sName` is the parameter's name; `variantValues` is an array of value and datatype-value pairs.
- For more information and code samples, see the section, XBRL Formula Parameters.

`void` **`AddFormulaParameter(`**`string sType, string sName, string sValue, string sNamespace`**`)` [Top | Methods | Properties]

- Adds a parameter for formula evaluation. It is deprecated.
- All the arguments are strings: `sType` is the datatype of the parameter; `sName` is the parameter's name; `sValue` is the parameter value; and `sNamespace` is the parameter's namespace.
- Each parameter must be specified in a separate call to the method.

`void` **`AddFormulaParameter(`**`string sType, string sName, string sValue, string sNamespace = ""`**`)` [Top | Methods | Properties]

- Adds a parameter for formula evaluation.
- All the arguments are strings: `sType` is the datatype of the parameter; `sName` is the parameter's name; `sValue` is the parameter value; and `sNamespace` is the parameter's namespace and is the empty string.
- Each parameter must be specified in a separate call to the method.

`void` **`AddFormulaParameterNamespace(`**`string sPrefix, string sURI`**`)` [Top | Methods | Properties]

- Defines a namespace used in the QNames of parameter names, types, or values.
- All the arguments are strings: `sPrefix` is the The namespace-prefix of values passed to `AddFormulaArrayParameter`; `sURI` is the namespace URI.
- Each parameter must be specified in a separate call to the method.

`void` **`ClearFormulaParameterList()`** [Top | Methods | Properties]

- Clears the list of formula parameters created with the `AddFormulaParameter` method.

`string` **`ReadFormulaAssertions()`** [Top | Methods | Properties]

- Reads formula assertions from the file being evaluated.

`string` **`ReadFormulaOutput()`** [Top | Methods | Properties]

- Reads the output of the file's formula assertions.

Properties

The properties of the **`IXBRL`** interface are described below in alphabetical order. The table arranges the properties in groups for ease of reference. Note that string inputs to be interpreted as URLs must provide absolute paths. If a relative path is used, a mechanism to resolve the relative path should be defined in the calling module.

`string` **`AddAssertionForProcessing`** [Top | Methods | Properties]

Limits assertion evaluation to the given assertion only. Call multiple times to specify more than one assertion. Use `##none` for no assertion, and `##all` for all assertions.

`string` **`AddAssertionSetForProcessing`** [Top | Methods | Properties]

Limits assertion set evaluation to the given assertion set only. Call multiple times to specify more than one assertion set. Use `##none` for no assertion set, and `##all` for all assertion sets.

`string` **AddTableForProcessing** [Top | Methods | Properties]
Limits table generation to the given table only. Call multiple times to specify more than one table. Use `##none` for no table, and `##all` for all tables.

`string` **ConceptLabelLinkrole** [Top | Methods | Properties]
Specifies the preferred extended link role to use when rendering concept labels.

`string` **ConceptLabelRole** [Top | Methods | Properties]
Specifies the preferred label role to use when rendering concept labels. Default is: `http://www.xbrl.org/2008/role/label`.

`bool` **DimensionExtensionEnabled** [Top | Methods | Properties]
Enables or disables XBRL dimension extensions validation. A value of `true` enables dimension extensions validation; `false` disables it. Default is `true`.

`bool` **EvaluateReferencedParametersOnly** [Top | Methods | Properties]
If `false`, forces evaluation of all parameters even if they are not referenced by any formulas/assertions/tables. Default is: `true`.

`bool` **FormulaAssertionsAsXML** [Top | Methods | Properties]
Enables XML formatting of the formula assertions file when RaptorXML is run with assertions enabled. A value of `true` enables XML formating; a value of `false` generates JSON output. Default is `false`.

`string` **FormulaAssertionsOutput** [Top | Methods | Properties]
Specifies the location of the formula assertion output file. The full path must be specified.

`bool` **FormulaExtensionEnabled** [Top | Methods | Properties]
Enables or disables XBRL formula extensions validation. A value of `true` enables formula extensions validation; `false` disables it. Default is `true`.

`string` **FormulaOutput** [Top | Methods | Properties]
Specifies the location of the output of the XBRL formula evaluation file. The full path must be specified.

string **FormulaParameterFile** [Top | Methods | Properties]
Specifies the location of the formula parameter file. The full path must be specified.

bool **FormulaPreloadSchemas** [Top | Methods | Properties]
Defines whether the formula schemas will be preloaded. A value of `true` preloads the schemas. The default is `false`, which causes these schemas not to be preloaded.

string **GenericLabelLinkrole** [Top | Methods | Properties]
Specifies the preferred extended link role to use when rendering generic labels.

string **GenericLabelRole** [Top | Methods | Properties]
Specifies the preferred label role to use when rendering generic labels. Default is: `http://www.xbrl.org/2008/role/label`.

object **InputFileArray** [Top | Methods | Properties]
Sets the array of XBRL files that will be used as input data/instances. The array is an object containing the strings of the absolute URLs of each of the input files.

string **InputFileName** [Top | Methods | Properties]
Specifies the filename and location of the XBRL instance file. The submitted string must be either an absolute URL; relative paths can be resolved relative to a base location, according to a mechanism defined in the calling module.

string **InputFromText** [Top | Methods | Properties]
Supplies the contents of the XBRL input document as text.

object **InputTextArray** [Top | Methods | Properties]
Sets the array of text files that will be used as input data. The array is an object containing the strings of the absolute URLs of each of the input files.

string **LabelLang** [Top | Methods | Properties]
Specifies the preferred label language to use when rendering labels. Default is: `en`.

string **LastErrorMessage** [Top | Methods | Properties]
Retrieves the last error message from the RaptorXML Engine as a string.

bool **ParallelAssessment** [Top | Methods | Properties]
Enables/disables parallel schema validity assessment.

bool **PreloadSchemas** [Top | Methods | Properties]
Defines whether the XBRL 2.1 schemas will be preloaded. A value of `true` preloads the schemas. The default is `true`.

string **PythonScriptFile** [Top | Methods | Properties]
Specifies the Python script file that provides additional processing of the XML or XSD file submitted for validation. The supplied string must be an absolute URL that gives the base location of the Python script.

ENUMSchemaImports **SchemaImports** [Top | Methods | Properties]
Specifies how schema imports are to be handled according to the attribute values of the `xs:import` elements. The handling is specified by the ENUMSchemaImports literal that is selected.

ENUMLoadSchemalocation **SchemalocationHints** [Top | Methods | Properties]
Specifies the mechanism to use to locate the schema. The mechanism is specified by the ENUMLoadSchemalocation literal that is selected.

ENUMSchemaMapping **SchemaMapping** [Top | Methods | Properties]
Sets what mapping to use in order to locate the schema. The mapping is specified by the ENUMSchemaMapping literal that is selected.

bool **TableEliminateEmptyRows** [Top | Methods | Properties]
Enables the elimination of empty rows/columns in the HTML output of table generation.

bool **TableExtensionEnabled** [Top | Methods | Properties]
Enables/disables the XBRL Table 1.0 extension.

string **TableLinkbaseNamespace** [Top | Methods | Properties]
Enables the loading of table linkbases written with a previous draft specification. The supplied string value specifies the table linkbase. Table linkbase validation, resolution, and layout is, however, always performed according to the Table Linkbase 1.0 Recommendation of 18 March 2014. Use `##detect` to enable auto-detection. The following values are recognized:

```
##detect
```



```
http://xbrl.org/PWD/2013-05-17/table
http://xbrl.org/PWD/2013-08-28/table
http://xbrl.org/CR/2013-11-13/table
http://xbrl.org/PR/2013-12-18/table
http://xbrl.org/2014/table
```

`string` **TableOutput** [Top | Methods | Properties]
Specifies the filename and location of the output of table generation. The submitted string must be the full path of the output file.

ENUMTableOutputFormat **TableOutputFormat** [Top | Methods | Properties]
Specifies the format of the table-generation output file.

`bool` **TablePreloadSchemas** [Top | Methods | Properties]
Enables/disables preloading of the XBRL Table 1.0 specification schemas.

`bool` **TreatXBRLInconsistenciesAsErrors** [Top | Methods | Properties]
A value of `true` causes XBRL validation to fail if the file contains any inconsistencies as defined by the XBRL 2.1 specification. Default is `false`: XBRL inconsistencies according to the XBRL 2.1 specification are not treated as errors.

`bool` **XincludeSupport** [Top | Methods | Properties]
Enables the use of `XInclude` elements. A value of `true` enables XInclude support; `false` disables it.

7.4.2 Enumerations

The following enumerations are defined. They are described in the sub-sections of this section.

ENUMAssessmentMode

ENUMErrorFormat

ENUMLoadSchemalocation

ENUMQueryVersion

ENUMSchemaImports

ENUMSchemaMapping

ENUMValidationType

ENUMWellformedCheckType

ENUMXBRLValidationType

ENUMXMLValidationMode

ENUMXQueryVersion

ENUMXSDVersion

ENUMXSLTVersion

ENUMAssessmentMode

Description

Contains enumeration literals that define the assessment mode of the XML Validator: `Strict` or `Lax`.

Used by

Interface	*Operation*
IXMLValidator	AssessmentMode

Enumeration literals

```
eAssessmentModeStrict   = 0
eAssessmentModeLax      = 1
```

`eAssessmentModeStrict`

Sets the schema-validity assessment mode to `Strict`. This is the default value.

eAssessmentModeLax
Sets the schema-validity assessment mode to `Lax`.

ENUMErrorFormat

Description
Contains enumeration literals specifying the format of error output.

Used by

Interface	*Operation*
IServer	ErrorFormat

Enumeration literals

```
eFormatText          = 0
eFormatShortXML      = 1
eFormatLongXML       = 2
```

eFormatText
Sets the error output format to `Text`. The default value.

eFormatShortXML
Sets the error output format to `ShortXML`. This format is an abbreviated form of the `LongXML` format.

eFormatLongXML
Sets the error output format to `LongXML`. This format provides the most detail of all three output formats.

ENUMLoadSchemalocation

Description
Contains enumeration literals that indicate how the schema's location should be determined.

Used by

Interface	*Operation*
IXBRL	SchemalocationHints
IXMLValidator	SchemalocationHints
IXSLT	SchemalocationHints

Enumeration literals

```
eSHLoadBySchemalocation     = 0
eSHLoadByNamespace          = 1
eSHLoadCombiningBoth        = 2
eSHLoadIgnore               = 3
```

eSHLoadBySchemalocation
Sets Load Schemalocation 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**.

eSHLoadByNamespace
Sets Load Schemalocation to `LoadByNamespace`. Uses the namespace part of `xsi:schemaLocation` (an empty string in the case of `xsi:noNamespaceSchemaLocation`), and locates the schema via a catalog mapping.

eSHLoadCombiningBoth
Sets Load Schemalocation 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 `ENUMSchemaMapping` parameter decides which mapping is used. If neither the namespace nor URL has a catalog mapping, the URL is used.

eSHLoadIgnore
Sets Load Schemalocation to `LoadIgnore`. If the parameter's value is `eSHLoadIgnore`, then the `xsi:schemaLocation` and `xsi:noNamespaceSchemaLocation` attributes are both ignored.

ENUMQueryVersion

Description
Contains enumeration literals that specify the XQuery version to use: XQuery 1.0 or 3.0.

Enumeration literals

```
eXQVersion10        = 1
eXQVersion30        = 3
```

eXQVersion10
Sets the XQuery version to XQuery 1.0.

eXQVersion30
Sets the XQuery version to XQuery 3.0.

ENUMSchemaImports

Description
Contains the enumeration literals that define the behaviour of `xs:import` elements. The `xs:import` element has `namespace` and `schemaLocation` attributes, both optional.

Used by

Interface	*Operation*
IXBRL	SchemaImports
IXMLValidator	SchemaImports
IXSLT	SchemaImports

Enumeration literals

```
eSILoadBySchemalocation          = 0
eSILoadPreferringSchemalocation  = 1
eSILoadByNamespace               = 2
eSICombiningBoth                 = 3
eSILicenseNamespaceOnly          = 4
```

eSILoadBySchemalocation
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).

eSILoadPreferringSchemalocation
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 literal is the **default value** of the enumeration.

eSILoadByNamespace
Sets the Schema Import to `LoadByNamespace`. The value of the `namespace` attribute is used to locate the schema via a catalog mapping.

eSICombiningBoth

Sets the Schema Import to `CombiningBoth`. If either the `namespace` or `schemaLocation` attribute has a catalog mapping, then that catalog mapping is used. If both have catalog mappings, then the value of the `ENUMSchemaMapping` parameter decides which mapping is used. If no catalog mapping is present, the value of the `schemaLocation` attribute (which should be a URL) is used.

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

ENUMSchemaMapping

Description
Contains the enumeration literals that define which of two catalog mappings is preferred: namespaces or schema-location URLs. This enumeration is useful for disambiguating `ENUMLoadSchemalocation` and `ENUMSchemaImports`.

Used by

Interface	*Operation*
IXBRL	SchemaMapping
IXMLValidator	SchemaMapping
IXSLT	SchemaMapping

Enumeration literals

```
eSMPreferSchemalocation     = 0
eSMPreferNamespace          = 1
```

eSMPreferSchemalocation
Sets the schema mapping option to select the schema location URL.

eSMPreferNamespace
Sets the schema mapping option to select the namespace.

ENUMTableOutputFormat

Description
Contains the enumeration literals that that specifies the output format of the document containing the generated tables.

Used by

Interface	*Operation*

IXBRL	TableOutputFormat

Enumeration literals

```
eFormatXML                  = 0
eFormatHTML                 = 1
```

eSMPreferSchemalocation
Sets the schema mapping option to select the schema location URL.

eSMPreferNamespace
Sets the schema mapping option to select the namespace.

ENUMValidationType

Description
Contains enumeration literals that define the type of document to validate.

Used by

Interface	*Operation*
IXMLValidator	IsValid

Enumeration literals

```
eValidateAny        = 0
eValidateXMLWithDTD = 1
eValidateXMLWithXSD = 2
eValidateDTD        = 3
eValidateXSD        = 4
```

eValidateAny
Sets the validation type to `Any`. This validates a document after automatically detecting its type.

eValidateXMLWithDTD
Sets the validation type to `XMLWithDTD`. This specifies validation of an XML document against a DTD.

eValidateXMLWithXSD
Sets the validation type to `XMLWithXSD`. This specifies validation of an XML document against an XML Schema.

eValidateDTD
Sets the validation type to `ValidateDTD`. This specifies validation of a DTD document.

eValidateXSD
Sets the validation type to `ValidateXSD`. This specifies validation of a W3C XML Schema document.

ENUMWellformedCheckType

Description
Contains the enumeration literals that define the type of document to check: XML or DTD.

Used by

Interface	*Operation*
`IXMLValidator`	`IsWellFormed`

Enumeration literals

```
eWellFormedAny      = 0
eWellFormedXML      = 1
eWellFormedDTD      = 2
```

eWellformedAny
Sets the well-formed check type to `Any`. This checks an XML or DTD document for well-formedness after automatically detecting which of the two types it is.

eWellformedXML
Sets the well-formed check type to `XML`. This checks an XML document for well-formedness according to the XML 1.0 or XML 1.1 specification.

eWellformedDTD
Sets the well-formed check type to `DTD`. This checks a DTD document for well-formedness.

ENUMXBRLValidationType

Description

Contains enumeration literals that define the type of XBRL document to validate: XBRL instance or XBRL taxonomy.

Used by

Interface	*Operation*
IXBRL	IsValid

Enumeration literals

```
eValidateXBRLAny          = 0
eValidateXBRLInstance     = 1
eValidateXBRLTaxonomy     = 2
```

eValidateXBRLAny

Sets the validation type to `Any`. This validates the XBRL document after detecting its type (instance or taxonomy) automatically.

eValidateXBRLInstance

Sets the validation type to `Instance`. This specifies validation of one or more XBRL instance documents.

eValidateXBRLTaxonomy

Sets the validation type to `Taxonomy`. This specifies validation of one or more XBRL taxonomy documents.

ENUMXMLValidationMode

Description

Contains the enumeration literals that define the XML processing mode to use: Validation or Wellformed.

Used by

Interface	*Operation*
IXMLValidator	XMLValidationMode
IXQuery	XMLValidationMode
IXSLT	XMLValidationMode

Enumeration literals

```
eXMLValidationModeWF      = 0
eXMLValidationModeID      = 1
eXMLValidationModeValid   = 2
```

eXMLValidationModeWF
Sets the XML processing mode to `Wellformed`. This is the default value.

eXMLValidationModeID
Internal.

eXMLValidationModeValid
Sets the XML processing mode to `Validation`.

ENUMXQueryVersion

Description
Contains enumeration literals that specify the XQuery version to use: XQuery 1.0 or 3.0.

Used by

Interface	*Operation*
IXQuery	EngineVersion

Enumeration literals

```
eXQVersion10        = 1
eXQVersion30        = 3
```

eXQVersion10
Sets the XQuery version to XQuery 1.0.

eXQVersion30
Sets the XQuery version to XQuery 3.0. This is the default value.

ENUMXQueryUpdatedXML

Description
Contains enumeration literals to specify how XQuery updates are handled.

Used by

Interface	*Operation*
IXQuery	UpdatedXMLWriteMode

Enumeration literals

```
eUpdatedDiscard          = 1
eUpdatedWriteback        = 2
eUpdatedAsMainResult     = 3
```

eUpdatedDiscard
Updates are discarded and not written to file.

eUpdatedWriteback
Updates are written to the input XML file specified with InputXMLFileName.

eUpdatedAsMainResult
Updates are written to the location specified by the `outputFile` parameter of ExecuteUpdate .

ENUMXSDVersion

Description
Contains enumeration literals that indicate the XML Schema version to use for validation: XSD 1.0 or 1.1.

Used by

Interface	*Operation*
IXMLValidator	XSDVersion
IXQuery	XSDVersion
IXSLT	XSDVersion

Enumeration literals

```
eXSDVersionAuto      = 0
eXSDVersion10        = 1
eXSDVersion11        = 2
```

eXSDVersionAuto
Sets the XML Schema version for validation to `Auto-detect`. The XSD version will be detected automatically after parsing the XSD document. If the XSD document's `vc:minVersion` attribute has a value of `1.1`, the document will be considered to be XSD 1.1. If the attribute has any other value, or does not exist, the document will be considered to be XSD 1.0.

eXSDVersion10
Sets the XML Schema version for validation to XML Schema 1.0.

eXSDVersion11
Sets the XML Schema version for validation to XML-Schema 1.1.

ENUMXSLTVersion

Description
Contains enumeration literals that define the XSLT version to use: XSLT 1.0, 2.0, or 3.0.

Used by

Interface	*Operation*
IXSLT	EngineVersion

Enumeration literals

```
eVersion10     = 1
eVersion20     = 2
eVersion30     = 3
```

eVersion10
Sets the XSLT version to XSLT 1.0.

eVersion20
Sets the XSLT version to XSLT 2.0.

eVersion30
Sets the XSLT version to XSLT 3.0.

Chapter 8

Additional Information

8 Additional Information

This section contains the following additional information:

- XBRL Formula Parameter

8.1 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 (for XBRL and XSD/XML) specifies how the schema location is to be selected.

8.2 XBRL Formula Parameters

This section contains the following topics:

- Formula Parameter Formats, which gives examples of the XML and JSON formats of XBRL formula parameters.
- Using Formula Parameters contains listings in Java, VB.NET, C#, VBScript, and JScript that show formula parameters can be used using objects from the Java and COM/.NET API libraries.

8.2.1 Formula Parameter Formats

Formula parameters can be given in XML format or JSON format.

XML format
The listing below shows formula parameters in XML format.

```
<?xml version="1.0" encoding="utf-8"?>
<options:formula-parameters
  xmlns:options="http://www.altova.com/schemas/altova/raptorxml/options"
  xmlns:p="http://xbrl.org/formula/conformance/paramstuff"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.altova.com/schemas/altova/raptorxml/options
http://www.altova.com/schemas/altova/raptorxml/options.xsd">

  <options:parameter name="p1">
    <options:value type="xs:string">hello world from new xml (without namespace)
</options:value>
  </options:parameter>
  <options:parameter name="p:p1" type="xs:string" value="hello world from new
xml"/>

</options:formula-parameters>
```

Note the following points:

- The `@type` attribute is optional and defaults to `xs:string`.
- Multiple `<options:value>` child elements can be specified in order to assign an XPath sequence to a parameter.
- `@value` and `<options:value>` cannot be used at the same time.

JSON format
The listing below shows formula parameters in JSON format.

```
{
  "formula-parameters": [
  {
    "name": "p1",
    "values": [
      {
        "type": "xs:string",
        "value": "hello world from json new (without namespace)"
      }
    ]
   }, {
        "name": "ns1:p1",
        "values": [
```

```
            {
              "type": "xs:string",
              "value": "hello world from json new"
            }
          ]
        }
    ],
    "namespaces": {
      "xs": "http://www.w3.org/2001/XMLSchema",
      "ns1": "http://xbrl.org/formula/conformance/paramstuff"
    }
}
```

Note the following points:

- The `type` key is optional and defaults to `xs:string`.
- The `xs` key is optional and defaults to `http://www.w3.org/2001/XMLSchema`.
- The type in the parameter map is used if a value is specified directly as a JSON string.
- Other ways of writing are currently also supported:

```
{
    "name": "p2",
    "type": "xs:string",
    "value": "hello world from json new (without namespace)"
}, {
    "name": "p3",
    "type": "xs:int",
    "values": ["1", "2"]
}, {
    "name": "p4",
    "type": "xs: int",
    "values": ["1", {"type": "xs:string", "value": "abc"}, "2"]
}
```


8.2.2 Using Formula Parameters

The example listings below shows how XBRL formula parameters can be used in various programming languages. For Java, see the Java API's XBRL class. For the other languages, refer to the COM/.NET API's XBRL interface.

Java

```
RaptorXMLFactory rxml = RaptorXML.getFactory();
XBRL xbrl = rxml.getXBRL();

xbrl.addFormulaParameter( "ns1:string", "ns1:Param1", "ns1:theqname" );
xbrl.addFormulaParameterNamespace( "ns1", "www.www.www" );

// The parameter is an array of dates
xbrl.addFormulaArrayParameter( "", "startDates", new Object[]{ new
FormulaParam( "xs:date", "2010-01-01" ), new FormulaParam( "xs:date",
"2012-01-01" ) } );

// The parameter is an array of figs
xbrl.addFormulaArrayParameter( "ns1:figs", "startFigs", new Object[]
{ "fig1", "fig2", "fig3" } );

// The parameter is an array of figs, dates and raisins (rather wild
example)
xbrl.addFormulaArrayParameter( "ns1:figs", "startDryFruit", new Object[]
{ "fig1", "fig2", new FormulaParam( "xs:date", "2010-01-01" ), new
FormulaParam( "ns1:raisin", "dried grape" ), "fig3" } );
```

VB.NET

```
Dim objRaptor As New Server()
   Dim objXBRL As XBRL
   objXBRL = objRaptor.GetXBRL()

   objXBRL.AddFormulaParameter("ns1:string", "ns1:Param1", "ns1:theqname")
   objXBRL.AddFormulaParameterNamespace("ns1", "www.www.www")

   'The parameter is an array of dates
   objXBRL.AddFormulaArrayParameter("", "startDates", {New XBRLFormulaParam
 With {.ParamType = "xs:date", .ParamValue = "2010-01-01"}, New
XBRLFormulaParam With {.ParamType = "xs:date", .ParamValue = "2012-01-
01"}})

   'The parameter is an array of figs
   objXBRL.AddFormulaArrayParameter("ns1:figs", "startFigs", {"fig1",
"fig2", "fig3"})

   'The parameter is an array of figs, dates and raisins (rather wild
example)
   objXBRL.AddFormulaArrayParameter("ns1:figs", "startDryFruit", {"fig1",
"fig2", New XBRLFormulaParam With {.ParamType = "xs:date", .ParamValue =
```

```
"2010-01-01"}, New XBRLFormulaParam With {.ParamType =
"ns1:raisin", .ParamValue = "dried grape"}, "fig3"})
```

C#

```
Server app = new Server();
   XBRL objXBRL = app.GetXBRL();

   objXBRL.AddFormulaParameter("ns1:string", "ns1:Param1", "ns1:theqname");
   objXBRL.AddFormulaParameterNamespace("ns1", "www.www.www");

   //The parameter is an array of dates
   objXBRL.AddFormulaArrayParameter("", "startDates", new object[] {new
XBRLFormulaParam { ParamType = "xs:date", ParamValue = "2010-01-01"}, new
XBRLFormulaParam {ParamType = "xs:date", ParamValue = "2012-01-01"}});

   //The parameter is an array of figs
   objXBRL.AddFormulaArrayParameter("ns1:figs", "startFigs", new object[]
{"fig1", "fig2", "fig3"});

   //The parameter is an array of figs, dates and raisins (rather wild
example)
   objXBRL.AddFormulaArrayParameter("ns1:figs", "startDryFruit", new
object[] { "fig1", "fig2", new XBRLFormulaParam { ParamType = "xs:date",
ParamValue = "2010-01-01" }, new XBRLFormulaParam { ParamType =
"ns1:raisin", ParamValue = "dried grape" }, "fig3" });
```

VBScript

Since the Raptor type library cannot be loaded by scripting languages, and because the type `XBRLFormulaParameters` doesn't exist, the VBScript user, instead of using `XBRL.FormulaParam` objects, must declare a class in his/her program. The class *must* have two public properties, `ParamName` and `ParamValue` (just as the `XBRL.FormulaParam` has). The class should have a constructor that takes the type and value, since this simplifies usage; otherwise the object needs to be created and have its members set). See the COM/.NET API's XBRL interface.

```
Class MyPair
   Public ParamType
   Public ParamValue
   Public Default Function Init( inType, inValue )
          ParamType = inType
          ParamValue = inValue
          set Init = Me
   End Function
End Class
```



```
Sub Main
   Dim objRaptor
   Set objRaptor = WScript.GetObject( "", "RaptorXML.Server" )
   Dim objXBRL
   Set objXBRL = objRaptor.GetXBRL

   Call objXBRL.AddFormulaParameter("ns1:string", "ns1:Param1",
"ns1:theqname")
   Call objXBRL.AddFormulaParameterNamespace("ns1", "www.www.www")

   'The parameter is an array of dates
   Call objXBRL.AddFormulaArrayParameter("", "startDates", Array( ( New
MyPair)( "xs:date", "2010-01-01"), ( New MyPair)( "xs:date", "2012-01-
01") ) )

   'The parameter is an array of figs
   Call objXBRL.AddFormulaArrayParameter("ns1:figs", "startFigs",
Array("fig1", "fig2", "fig3") )

   'The parameter is an array of figs, dates and raisins (rather wild
example)
   Call objXBRL.AddFormulaArrayParameter("ns1:figs", "startDryFruit",
Array("fig1", "fig2", ( New MyPair)("xs:date", "2010-01-01"(, ( New MyPair)
("ns1:raisin", "dried grape"), "fig3") )
End Sub

Call Main
```

JScript

Since the Raptor type library cannot be loaded by scripting languages, and because the type `XBRLFormulaParameters` doesn't exist, the JScript user, instead of using `XBRL.FormulaParam` objects, must declare function-classes in his/her program that holds the type-value pair. Names of members *must* be ParamType and `ParamValue`. See the COM/.NET API's XBRL interface.

```
function FormulaParam( inType, inValue)
{
   this.ParamType = inType;
   this.ParamValue = inValue;
}

function main()
{
   var objRaptor = new ActiveXObject( "RaptorXML.Server" );
   var objXBRL = objRaptor.GetXBRL();

   objXBRL.addFormulaParameter( "ns1:string", "ns1:Param1",
"ns1:theqname" );
   objXBRL.addFormulaParameter( "xs:string", "Param1", "bla",
```

```
"www.www.www" );

   // The parameter is an array of dates
   objXBRL.addFormulaArrayParameter("", "startDates", [new
FormulaParam("xs:date", "2010-01-01"), new FormulaParam("xs:date", "2012-
01-01")]);

   // The parameter is an array of figs
   objXBRL.addFormulaArrayParameter("ns1:figs", "startFigs", ["fig1",
"fig2", "fig3"]);

   // The parameter is an array of figs, dates and raisins (rather wild
example)
   objXBRL.addFormulaArrayParameter("ns1:figs", "startDryFruit", ["fig1",
"fig2", new FormulaParam("xs:date", "2010-01-01"), new
FormulaParam("ns1:raisin", "dried grape"), "fig3"]);
}

main()
```


Chapter 9

XSLT and XQuery Engine Information

9 XSLT and XQuery Engine Information

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

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:

9.1 XSLT 1.0

The XSLT 1.0 Engine of RaptorXML+XBRL Server 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.

9.2 XSLT 2.0

This section:

Conformance
The XSLT 2.0 engine of RaptorXML+XBRL Server 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="http://www.w3.org/2005/xpath-functions"
    ...
</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`.

9.3 XSLT 3.0

The XSLT 3.0 Engine of RaptorXML+XBRL Server conforms to the World Wide Web Consortium's (W3C's) XSLT 3.0 Last Call Working Draft of 12 December 2013 and XPath 3.0 Recommendation of 8 April 2014.

The XSLT 3.0 engine has the same implementation-specific characteristics as the XSLT 2.0 engine. Additionally, it includes support for the following XSLT 3.0 features: `xsl:evaluate`, `xsl:try`, `xsl:catch`, `xsl:map`, `xsl:map-entry`, text value templates, XPath/XQuery 3.0 functions and operators, and the XPath 3.0 specification.

The following XSLT 3.0 instructions are currently unsupported:

```
xsl:accept
xsl:accumulator
xsl:accumulator-rule
xsl:assert
xsl:break
xsl:context-item
xsl:expose
xsl:fork
xsl:iterate
xsl:merge
xsl:merge-action
xsl:merge-key
xsl:merge-source
xsl:mode
xsl:next-iteration
xsl:next-match
xsl:on-completion
xsl:override
xsl:package
xsl:stream
xsl:use-package
```

9.4 XQuery 1.0

This section:

Conformance
The XQuery 1.0 Engine of RaptorXML+XBRL Server 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 23 January 2007, from the XPath Datatypes namespace to the XML Schema namespace, so: `xs:yearMonthDuration`.

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

XML source document and validation

XML documents used in executing an XQuery document with the 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.

9.5 XQuery 3.0

The XQuery 3.0 Engine of RaptorXML+XBRL Server conforms to the World Wide Web Consortium's (W3C's) XQuery 3.0 Recommendation of 8 April 2014 and includes support for XPath and XQuery Functions 3.0.

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

Chapter 10

XSLT and XPath/XQuery Functions

10 XSLT and XPath/XQuery Functions

This section lists Altova extension functions and other extension functions that can be used in XPath and/or XQuery expressions. Altova extension functions can be used with Altova's XSLT and XQuery engines, and provide functionality additional to that available in the function libraries defined in the W3C standards.

General points
The following general points should be noted:

- Functions from the core function libraries defined in the W3C specifications can be called without a prefix. That's because the XSLT and XQuery engines read non-prefixed functions as belonging to a default functions namespace which is that specified in the XPath/XQuery functions specifications `http://www.w3.org/2005/xpath-functions`. If this namespace is explicitly declared in an XSLT or XQuery document, the prefix used in the namespace declaration can also optionally be used on function names.
- 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 `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 (*table below*). Any string comparisons, including for the `max` and `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 `in-scope-prefixes()`, `namespace-uri()` and `namespace-uri-for-prefix()` functions.

10.1 Altova Extension Functions

Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT 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 this namespace. Note that, in future versions of your product, support for a function 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.

Functions defined in the W3C's XPath/XQuery Functions specifications can be used in: (i) XPath expressions in an XSLT context, and (ii) in XQuery expressions in an XQuery document. In this documentation we indicate the functions that can be used in the former context (XPath in XSLT) with an `XP` symbol and call them XPath functions; those functions that can be used in the latter (XQuery) context are indicated with an `XQ` symbol; they work as XQuery functions. The W3C's XSLT specifications—not XPath/XQuery Functions specifications—also define functions that can be used in XPath expressions in XSLT documents. These functions are marked with an `XSLT` symbol and are called XSLT functions. The XPath/XQuery and XSLT versions in which a function can be used are indicated in the description of the function (*see symbols below*). Functions from the XPath/XQuery and XSLT function libraries are listed without a prefix. Extension functions from other libraries, such as Altova extension functions, are listed with a prefix.

XPath functions (used in XPath expressions in XSLT):	`XP1` `XP2` `XP3`
XSLT functions (used in XPath expressions in XSLT):	`XSLT1` `XSLT2` `XSLT3`
XQuery functions (used in XQuery expressions in XQuery):	`XQ1` `XQ3`

XSLT functions
XSLT functions can only be used in XPath expressions in an XSLT context (similarly to XSLT 2.0's `current-group()` or `key()` functions). These functions are not intended for, and will not work in, a non-XSLT context (for instance, in an XQuery context). Note that XSLT functions for XBRL can be used only with editions of Altova products that have XBRL support.

XPath/XQuery functions
XPath/XQuery functions (general, date/time, and string) can be used both in XPath expressions in XSLT contexts as well as in XQuery expressions.

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.

Barcode functions
Altova's barcode extension functions enable barcodes to be generated and placed in output

generated via XSLT stylesheets.

10.1.1 XSLT Functions

XSLT extension functions can be used in XPath expressions in an XSLT context. They will not work in a non-XSLT context (for instance, in an XQuery context).

Note about naming of functions and language applicability
Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT 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 this namespace. Note that, in future versions of your product, support for a function 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.

XPath functions (used in XPath expressions in XSLT):	XP1 XP2 XP3
XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3
XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3

Standard functions

▼ **distinct-nodes [altova:]**

altova:distinct-nodes(*node()**) as **node()*** XSLT1 XSLT2 XSLT3

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`.

⊟ *Examples*

- **altova:distinct-nodes**(`country`) returns all child `country` nodes less those having duplicate values.

▼ **evaluate [altova:]**

altova:evaluate(XPathExpression *as xs:string***[, ValueOf$p1, ... ValueOf$pN])**
XSLT1 XSLT2 XSLT3

Takes an XPath expression, passed as a string, as its mandatory argument. It returns the output of the evaluated expression. For example: **altova:evaluate('//Name[1]')** returns the contents of the first `Name` element in the document. Note that the expression `//Name[1]` is passed as a string by enclosing it in single quotes.

The `altova:evaluate` function can optionally take additional arguments. These arguments are the values of in-scope variables that have the names `p1`, `p2`, `p3`... `pN`. Note the following points about usage: (i) The variables must be defined with names of the form `pX`, where `X` is an integer; (ii) the `altova:evaluate` function's arguments (*see signature above*), from the second argument onwards, provide the values of the variables, with the sequence of the arguments corresponding to the numerically ordered sequence of variables: `p1` to `pN`: The second argument will be the value of the variable `p1`, the third argument that of the variable `p2`, and so on; (iii) The variable values must be of type `item*`.

⊟ *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 listing above, 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.

Examples to further illustrate the use of variables

- ```
  <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`).

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

More examples

- 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.

encode-for-rtf [altova:]

altova:encode-for-rtf(input *as xs:string*, **preserveallwhitespace** *as xs:boolean*, **preservenewlines** *as xs:boolean*) as **xs:string** XSLT2 XSLT3

Converts the input string into code for RTF. Whitespace and new lines will be preserved according to the boolean value specified for their respective arguments.

[Top]

XBRL functions

Altova XBRL functions can be used only with editions of Altova products that have XBRL support.

xbrl-footnotes [altova:]

altova:xbrl-footnotes(*node()***)** as **node()*** XSLT2 XSLT3

Takes a node as its input argument and returns the set of XBRL footnote nodes referenced by the input node.

xbrl-labels [altova:]

altova:xbrl-labels(*xs:QName*, *xs:string***)** as **node()*** XSLT2 XSLT3

Takes two input arguments: a node name and the taxonomy file location containing the node. The function returns the XBRL label nodes associated with the input node.

[Top]

10.1.2 XPath/XQuery Functions: Date and Time

Altova's date/time extension functions can be used in XPath and XQuery expressions and provide additional functionality for the processing of data held as XML Schema's various date and time datatypes. The functions in this section can be used with Altova's **XPath 3.0** and **XQuery 3.0** engines. They are available in XPath/XQuery contexts.

Note about naming of functions and language applicability
Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT 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 this namespace. Note that, in future versions of your product, support for a function 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.

XPath functions (used in XPath expressions in XSLT):	XP1 XP2 XP3
XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3
XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3

▼ Grouped by functionality

- Add duration to xs:dateTime and return xs:dateTime
- Add a duration to xs:date and return xs:date
- Add a duration to xs:time and return xs:time
- Remove timezone from functions that generate current date/time
- Return weekday as integer from date
- Return week number as integer from date
- Build date, time, or duration type from lexical components of each type
- Construct date, dateTime, or time type from string input
- Age-related functions

▼ Grouped alphabetically

altova:add-days-to-date
altova:add-days-to-dateTime
altova:add-hours-to-dateTime
altova:add-hours-to-time
altova:add-minutes-to-dateTime
altova:add-minutes-to-time
altova:add-months-to-date
altova:add-months-to-dateTime
altova:add-seconds-to-dateTime
altova:add-seconds-to-time
altova:add-years-to-date
altova:add-years-to-dateTime
altova:age
altova:age-details
altova:build-date
altova:build-duration
altova:build-time
altova:current-dateTime-no-TZ
altova:current-date-no-TZ
altova:current-time-no-TZ

[Top]

Add a duration to xs:dateTime XP3 XQ3

These functions add a duration to **xs:dateTime** and return **xs:dateTime**. The xs:dateTime type has a format of CCYY-MM-DDThh:mm:ss.sss. This is a concatenation of the xs:date and xs:time formats separated by the letter T. A timezone suffix+01:00 (for example) is optional.

add-years-to-dateTime [altova:]

altova:add-years-to-dateTime(DateTime *as xs:dateTime*, **Years** *as xs:integer*) as **xs:dateTime** XP3 XQ3

Adds a duration in years to an xs:dateTime (*see examples below*). The second argument is the number of years to be added to the xs:dateTime supplied as the first argument. The result is of type xs:dateTime.

Examples

- **altova:add-years-to-dateTime**(xs:dateTime("2014-01-15T14:00:00"), 10) returns 2024-01-15T14:00:00
- **altova:add-years-to-dateTime**(xs:dateTime("2014-01-15T14:00:00"), -4) returns 2010-01-15T14:00:00

add-months-to-dateTime [altova:]

altova:add-months-to-dateTime(DateTime *as xs:dateTime*, **Months** *as xs:integer*) as **xs:dateTime** XP3 XQ3

Adds a duration in months to an xs:dateTime (*see examples below*). The second argument is the number of months to be added to the xs:dateTime supplied as the first argument. The result is of type xs:dateTime.

Examples

- **altova:add-months-to-dateTime**(xs:dateTime("2014-01-15T14:00:00"), 10) returns 2014-11-15T14:00:00
- **altova:add-months-to-dateTime**(xs:dateTime("2014-01-15T14:00:00"), -2) returns 2013-11-15T14:00:00

add-days-to-dateTime [altova:]

altova:add-days-to-dateTime(DateTime *as xs:dateTime*, **Days** *as xs:integer*) as **xs:dateTime** XP3 XQ3

Adds a duration in days to an xs:dateTime (*see examples below*). The second argument is the number of days to be added to the xs:dateTime supplied as the first argument. The result is of type xs:dateTime.

Examples

- **altova:add-days-to-dateTime**(xs:dateTime("2014-01-15T14:00:00"), 10)

returns 2014-01-25T14:00:00

- altova:add-days-to-dateTime(xs:dateTime("2014-01-15T14:00:00"), -8) returns 2014-01-07T14:00:00

add-hours-to-dateTime [altova:]

altova:add-hours-to-dateTime(DateTime *as xs:dateTime*, Hours *as xs:integer*) as xs:dateTime XP3 XQ3

Adds a duration in hours to an xs:dateTime (*see examples below*). The second argument is the number of hours to be added to the xs:dateTime supplied as the first argument. The result is of type xs:dateTime.

Examples

- altova:add-hours-to-dateTime(xs:dateTime("2014-01-15T13:00:00"), 10) returns 2014-01-15T23:00:00
- altova:add-hours-to-dateTime(xs:dateTime("2014-01-15T13:00:00"), -8) returns 2014-01-15T05:00:00

add-minutes-to-dateTime [altova:]

altova:add-minutes-to-dateTime(DateTime *as xs:dateTime*, Minutes *as xs:integer*) as xs:dateTime XP3 XQ3

Adds a duration in minutes to an xs:dateTime (*see examples below*). The second argument is the number of minutes to be added to the xs:dateTime supplied as the first argument. The result is of type xs:dateTime.

Examples

- altova:add-minutes-to-dateTime(xs:dateTime("2014-01-15T14:10:00"), 45) returns 2014-01-15T14:55:00
- altova:add-minutes-to-dateTime(xs:dateTime("2014-01-15T14:10:00"), -5) returns 2014-01-15T14:05:00

add-seconds-to-dateTime [altova:]

altova:add-seconds-to-dateTime(DateTime *as xs:dateTime*, Seconds *as xs:integer*) as xs:dateTime XP3 XQ3

Adds a duration in seconds to an xs:dateTime (*see examples below*). The second argument is the number of seconds to be added to the xs:dateTime supplied as the first argument. The result is of type xs:dateTime.

Examples

- altova:add-seconds-to-dateTime(xs:dateTime("2014-01-15T14:00:10"), 20) returns 2014-01-15T14:00:30
- altova:add-seconds-to-dateTime(xs:dateTime("2014-01-15T14:00:10"), -5) returns 2014-01-15T14:00:05

[Top]

Add a duration to xs:date XP3 XQ3

These functions add a duration to **xs:date** and return **xs:date**. The xs:date type has a format of

CCYY-MM-DD.

add-years-to-date [altova:]

altova:add-years-to-date(Date *as xs:date*, **Years** *as xs:integer*) as **xs:date**
XP3 XQ3
Adds a duration in years to a date. The second argument is the number of years to be added to the `xs:date` supplied as the first argument. The result is of type `xs:date`.

Examples

- **altova:add-years-to-date**(xs:date("2014-01-15"), 10) returns 2024-01-15
- **altova:add-years-to-date**(xs:date("2014-01-15"), -4) returns 2010-01-15

add-months-to-date [altova:]

altova:add-months-to-date(Date *as xs:date*, **Months** *as xs:integer*) as **xs:date**
XP3 XQ3
Adds a duration in months to a date. The second argument is the number of months to be added to the `xs:date` supplied as the first argument. The result is of type `xs:date`.

Examples

- **altova:add-months-to-date**(xs:date("2014-01-15"), 10) returns 2014-11-15
- **altova:add-months-to-date**(xs:date("2014-01-15"), -2) returns 2013-11-15

add-days-to-date [altova:]

altova:add-days-to-date(Date *as xs:date*, **Days** *as xs:integer*) as **xs:date** **XP3**
XQ3
Adds a duration in days to a date. The second argument is the number of days to be added to the `xs:date` supplied as the first argument. The result is of type `xs:date`.

Examples

- **altova:add-days-to-date**(xs:date("2014-01-15"), 10) returns 2014-01-25
- **altova:add-days-to-date**(xs:date("2014-01-15"), -8) returns 2014-01-07

[Top]

Add a duration to `xs:time` XP3 XQ3

These functions add a duration to **`xs:time`** and return **`xs:time`**. The `xs:time` type has a lexical form of `hh:mm:ss.sss`. An optional time zone may be suffixed. The letter `Z` indicates Coordinated Universal Time (UTC). All other time zones are represented by their difference from UTC in the format `+hh:mm`, or `-hh:mm`. If no time zone value is present, it is considered unknown; it is not assumed to be UTC.

add-hours-to-time [altova:]

altova:add-hours-to-time(Time *as xs:time*, **Hours** *as xs:integer*) as **xs:time**
XP3 XQ3
Adds a duration in hours to a time. The second argument is the number of hours to be added to the `xs:time` supplied as the first argument. The result is of type `xs:time`.

Examples

- **altova:add-hours-to-time**(xs:time("11:00:00"), 10) returns 21:00:00
- **altova:add-hours-to-time**(xs:time("11:00:00"), -7) returns 04:00:00

add-minutes-to-time [altova:]

altova:add-minutes-to-time(Time *as xs:time*, **Minutes** *as xs:integer*) as **xs:time**
XP3 XQ3

Adds a duration in minutes to a time. The second argument is the number of minutes to be added to the `xs:time` supplied as the first argument. The result is of type `xs:time`.

Examples

- **altova:add-minutes-to-time**(xs:time("14:10:00"), 45) returns 14:55:00
- **altova:add-minutes-to-time**(xs:time("14:10:00"), -5) returns 14:05:00

add-seconds-to-time [altova:]

altova:add-seconds-to-time(Time *as xs:time*, **Minutes** *as xs:integer*) as **xs:time**
XP3 XQ3

Adds a duration in seconds to a time. The second argument is the number of seconds to be added to the `xs:time` supplied as the first argument. The result is of type `xs:time`. The Seconds component can be in the range of `0` to `59.999`.

Examples

- **altova:add-seconds-to-time**(xs:time("14:00:00"), 20) returns 14:00:20
- **altova:add-seconds-to-time**(xs:time("14:00:00"), 20.895) returns 14:00:20.895

[Top]

Remove the timezone part from date/time datatypes **XP3 XQ3**

These functions remove the timezone from the current **xs:dateTime**, **xs:date**, or **xs:time** values, respectively. Note that the difference between `xs:dateTime` and `xs:dateTimeStamp` is that in the case of the latter the timezone part is required (while it is optional in the case of the former). So the format of an `xs:dateTimeStamp` value is: `CCYY-MM-DDThh:mm:ss.sss±hh:mm`. or `CCYY-MM-DDThh:mm:ss.sssZ`. If the date and time is read from the system clock as `xs:dateTimeStamp`, the `current-dateTime-no-TZ()` function can be used to remove the timezone if so required.

current-dateTime-no-TZ [altova:]

altova:current-dateTime-no-TZ() as **xs:dateTime** **XP3 XQ3**

This function takes no argument. It removes the timezone part of `current-dateTime()` (which is the current date-and-time according to the system clock) and returns an `xs:dateTime` value.

Examples

If the current dateTime is **2014-01-15T14:00:00+01:00**:

- **altova:current-dateTime-no-TZ**() returns 2014-01-15T14:00:00

current-date-no-TZ [altova:]

altova:current-date-no-TZ() as **xs:date** **XP3 XQ3**

This function takes no argument. It removes the timezone part of `current-date()` (which is the current date according to the system clock) and returns an `xs:date` value.

Examples

If the current date is `2014-01-15+01:00`:

- `altova:current-date-no-TZ()` returns `2014-01-15`

current-time-no-TZ [altova:]

`altova:current-time-no-TZ()` as `xs:time` XP3 XQ3

This function takes no argument. It removes the timezone part of `current-time()` (which is the current time according to the system clock) and returns an `xs:time` value.

Examples

If the current time is `14:00:00+01:00`:

- `altova:current-time-no-TZ()` returns `14:00:00`

[Top]

Return the weekday from `xs:dateTime` or `xs:date` XP3 XQ3

These functions return the weekday (as an integer) from `xs:dateTime` or `xs:date`. The days of the week are numbered (using the American format) from `1` to `7`, with `Sunday=1`. In the European format, the week starts with Monday (`=1`). The American format, where `Sunday=1`, can be set by using the integer `0` where an integer is accepted to indicate the format.

weekday-from-dateTime [altova:]

`altova:weekday-from-dateTime(DateTime` *as xs:dateTime*`)` as `xs:integer` XP3 XQ3

Takes a date-with-time as its single argument and returns the day of the week of this date as an integer. The weekdays are numbered starting with `Sunday=1`. If the European format is required (where `Monday=1`), use the other signature of this function (*see next signature below*).

Examples

- `altova:weekday-from-dateTime(xs:dateTime("2014-02-03T09:00:00"))` returns `2`, which would indicate a Monday.

`altova:weekday-from-dateTime(DateTime` *as xs:dateTime*`, Format` *as xs:integer*`)` as `xs:integer` XP3 XQ3

Takes a date-with-time as its first argument and returns the day of the week of this date as an integer. The weekdays are numbered starting with `Monday=1`. If the second (integer) argument is `0`, then the weekdays are numbered `1` to `7` starting with `Sunday=1`. If the second argument is an integer other than `0`, then `Monday=1`. If there is no second argument, the function is read as having the other signature of this function (*see previous signature*).

Examples

- `altova:weekday-from-dateTime(xs:dateTime("2014-02-03T09:00:00"), 1)` returns `1`, which would indicate a Monday
- `altova:weekday-from-dateTime(xs:dateTime("2014-02-03T09:00:00"), 4)` returns `1`, which would indicate a Monday
- `altova:weekday-from-dateTime(xs:dateTime("2014-02-03T09:00:00"), 0)` returns `2`, which would indicate a Monday.

weekday-from-date [altova:]

altova:weekday-from-date(Date *as xs:date***)** as **xs:integer** XP3 XQ3

Takes a date as its single argument and returns the day of the week of this date as an integer. The weekdays are numbered starting with `Sunday=1`. If the European format is required (where `Monday=1`), use the other signature of this function (*see next signature below*).

Examples

- **altova:weekday-from-date**(xs:date("2014-02-03+01:00")) returns 2, which would indicate a Monday.

altova:weekday-from-date(Date *as xs:date*, **Format** *as xs:integer***)** as **xs:integer** XP3 XQ3

Takes a date as its first argument and returns the day of the week of this date as an integer. The weekdays are numbered starting with `Monday=1`. If the second (`Format`) argument is `0`, then the weekdays are numbered `1` to `7` starting with `Sunday=1`. If the second argument is an integer other than `0`, then `Monday=1`. If there is no second argument, the function is read as having the other signature of this function (*see previous signature*).

Examples

- **altova:weekday-from-date**(xs:date("2014-02-03"), 1) returns 1, which would indicate a Monday
- **altova:weekday-from-date**(xs:date("2014-02-03"), 4) returns 1, which would indicate a Monday
- **altova:weekday-from-date**(xs:date("2014-02-03"), 0) returns 2, which would indicate a Monday.

[Top]

Return the week number from `xs:dateTime` or `xs:date` XP2 XQ1 XP3 XQ3

These functions return the week number (as an integer) from `xs:dateTime` or `xs:date`. Week-numbering is available in the US, ISO/European, and Islamic calendar formats. Week-numbering is different in these calendar formats because the week is considered to start on different days (on Sunday in the US format, Monday in the ISO/European format, and Saturday in the Islamic format).

weeknumber-from-date [altova:]

altova:weeknumber-from-date(Date *as xs:date*, **Calendar** *as xs:integer***)** as **xs:integer** XP2 XQ1 XP3 XQ3

Returns the week number of the submitted **`Date`** argument as an integer. The second argument (**`Calendar`**) specifies the calendar system to follow.

Supported **`Calendar`** values are:

- **`0 = US calendar`** (*week starts Sunday*)
- **`1 = ISO standard, European calendar`** (*week starts Monday*)
- **`2 = Islamic calendar`** (*week starts Saturday*)

Default is `0`.

Examples

- **altova:weeknumber-from-date**(xs:date("2014-03-23"), 0) returns 13
- **altova:weeknumber-from-date**(xs:date("2014-03-23"), 1) returns 12

- `altova:weeknumber-from-date(xs:date("2014-03-23"), 2)` returns `13`
- `altova:weeknumber-from-date(xs:date("2014-03-23") )` returns `13`

The day of the date in the examples above (`2014-03-23`) is Sunday. So the US and Islamic calendars are one week ahead of the European calendar on this day.

weeknumber-from-dateTime [altova:]

altova:weeknumber-from-dateTime(DateTime *as xs:dateTime*, **Calendar** *as xs:integer*) as **xs:integer** XP2 XQ1 XP3 XQ3

Returns the week number of the submitted **DateTime** argument as an integer. The second argument (**Calendar**) specifies the calendar system to follow.
Supported **Calendar** values are:

- **0 = US calendar** (*week starts Sunday*)
- **1 = ISO standard, European calendar** (*week starts Monday*)
- **2 = Islamic calendar** (*week starts Saturday*)

Default is **0**.

Examples

- `altova:weeknumber-from-dateTime(xs:dateTime("2014-03-23T00:00:00"), 0)` returns `13`
- `altova:weeknumber-from-dateTime(xs:dateTime("2014-03-23T00:00:00"), 1)` returns `12`
- `altova:weeknumber-from-dateTime(xs:dateTime("2014-03-23T00:00:00"), 2)` returns `13`
- `altova:weeknumber-from-dateTime(xs:dateTime("2014-03-23T00:00:00") )` returns `13`

The day of the dateTime in the examples above (`2014-03-23T00:00:00`) is Sunday. So the US and Islamic calendars are one week ahead of the European calendar on this day.

[Top]

Build date, time, and duration datatypes from their lexical components XP3 XQ3

The functions take the lexical components of the `xs:date`, `xs:time`, or `xs:duration` datatype as input arguments and combine them to build the respective datatype.

build-date [altova:]

altova:build-date(Year *as xs:integer*, **Month** *as xs:integer*, **Date** *as xs:integer*) as **xs:date** XP3 XQ3

The first, second, and third arguments are, respectively, the year, month, and date. They are combined to build a value of `xs:date` type. The values of the integers must be within the correct range of that particular date part. For example, the second argument (for the month part) should not be greater than `12`.

Examples

- `altova:build-date(2014, 2, 03)` returns `2014-02-03`

build-time [altova:]

altova:build-time(Hours *as xs:integer*, **Minutes** *as xs:integer*, **Seconds** *as xs:integer*) as **xs:time** **XP3 XQ3**

The first, second, and third arguments are, respectively, the hour (`0` to `23`), minutes (`0` to `59`), and seconds (`0` to `59`) values. They are combined to build a value of `xs:time` type. The values of the integers must be within the correct range of that particular time part. For example, the second (`Minutes`) argument should not be greater than `59`. To add a timezone part to the value, use the other signature of this function (*see next signature*).

Examples

- `altova:build-time(23, 4, 57)` returns `23:04:57`

altova:build-time(Hours *as xs:integer*, **Minutes** *as xs:integer*, **Seconds** *as xs:integer*, **TimeZone** *as xs:string*) as **xs:time** **XP3 XQ3**

The first, second, and third arguments are, respectively, the hour (`0` to `23`), minutes (`0` to `59`), and seconds (`0` to `59`) values. The fourth argument is a string that provides the timezone part of the value. The four arguments are combined to build a value of `xs:time` type. The values of the integers must be within the correct range of that particular time part. For example, the second (`Minutes`) argument should not be greater than `59`.

Examples

- `altova:build-time(23, 4, 57, '+1')` returns `23:04:57+01:00`

build-duration [altova:]

altova:build-duration(Years *as xs:integer*, **Months** *as xs:integer*) as **xs:yearMonthDuration** **XP3 XQ3**

Takes two arguments to build a value of type `xs:yearMonthDuration`. The first arguments provides the `Years` part of the duration value, while the second argument provides the `Months` part. If the second (`Months`) argument is greater than or equal to `12`, then the integer is divided by `12`; the quotient is added to the first argument to provide the `Years` part of the duration value while the remainder (of the division) provides the `Months` part. To build a duration of type `xs:dayTimeDuration`., see the next signature.

Examples

- `altova:build-duration(2, 10)` returns `P2Y10M`
- `altova:build-duration(14, 27)` returns `P16Y3M`
- `altova:build-duration(2, 24)` returns `P4Y`

altova:build-duration(Days *as xs:integer*, **Hours** *as xs:integer*, **Minutes** *as xs:integer*, **Seconds** *as xs:integer*) as **xs:dayTimeDuration** **XP3 XQ3**

Takes four arguments and combines them to build a value of type `xs:dayTimeDuration`. The first argument provides the `Days` part of the duration value, the second, third, and fourth arguments provide, respectively, the `Hours`, `Minutes`, and `Seconds` parts of the duration value. Each of the three Time arguments is converted to an equivalent value in terms of the next higher unit and the result is used for calculation of the total duration value. For example, 72 seconds is converted to `1M+12S` (1 minute and 12 seconds), and this value is used for calculation of the total duration value. To build a duration of type `xs:yearMonthDuration`., see the previous signature.

Examples

- `altova:build-duration(2, 10, 3, 56)` returns `P2DT10H3M56S`
- `altova:build-duration(1, 0, 100, 0)` returns `P1DT1H40M`
- `altova:build-duration(1, 0, 0, 3600)` returns `P1DT1H`

[Top]

Construct date, dateTime, and time datatypes from string input XP2 XQ1 XP3 XQ3

These functions take strings as arguments and construct `xs:date`, `xs:dateTime`, or `xs:time` datatypes. The string is analyzed for components of the datatype based on a submitted pattern argument.

parse-date [altova:]

altova:parse-date(Date *as xs:string*, **DatePattern** *as xs:string*) as **xs:date**
XP2 XQ1 XP3 XQ3

Returns the input string **Date** as an **xs:date** value. The second argument **DatePattern** specifies the pattern (sequence of components) of the input string. **DatePattern** is described with the component specifiers listed below and with component separators that can be any character. See the examples below.

D	Date
M	Month
Y	Year

The pattern in **DatePattern** must match the pattern in **Date**. Since the output is of type **xs:date**, the output will always have the lexical format **YYYY-MM-DD**.

Examples

- **altova:parse-date**(xs:string("06-03-2014"), "[D]-[M]-[Y]") returns 2014-03-06
- **altova:parse-date**(xs:string("06-03-2014"), "[M]-[D]-[Y]") returns 2014-06-03
- **altova:parse-date**("06/03/2014", "[M]/[D]/[Y]") returns 2014-06-03
- **altova:parse-date**("06 03 2014", "[M] [D] [Y]") returns 2014-06-03
- **altova:parse-date**("6 3 2014", "[M] [D] [Y]") returns 2014-06-03

parse-dateTime [altova:]

altova:parse-dateTime(DateTime *as xs:string*, **DateTimePattern** *as xs:string*) as **xs:dateTime** XP2 XQ1 XP3 XQ3

Returns the input string **DateTime** as an **xs:dateTime** value.The second argument **DateTimePattern** specifies the pattern (sequence of components) of the input string. **DateTimePattern** is described with the component specifiers listed below and with component separators that can be any character. See the examples below.

D	Date
M	Month
Y	Year
H	Hour
m	minutes
s	seconds

The pattern in **DateTimePattern** must match the pattern in **DateTime**. Since the output is of

type **xs:dateTime**, the output will always have the lexical format **YYYY-MM-DDTHH:mm:ss**.

Examples

- **altova:parse-dateTime**(xs:string("06-03-2014 13:56:24"), "[D]-[M]-[Y] [H]:[m]:[s]") returns 2014-03-06T13:56:24
- **altova:parse-dateTime**("time=13:56:24; date=06-03-2014", "time=[H]:[m]:[s]; date=[D]-[M]-[Y]") returns 2014-03-06T13:56:24

parse-time [altova:]

altova:parse-time(Time *as xs:string*, **TimePattern** *as xs:string***)** as **xs:time**
XP2 XQ1 XP3 XQ3

Returns the input string **Time** as an **xs:time** value.The second argument **TimePattern** specifies the pattern (sequence of components) of the input string. **TimePattern** is described with the component specifiers listed below and with component separators that can be any character. See the examples below.

H	Hour
m	minutes
s	seconds

The pattern in **TimePattern** must match the pattern in **Time**. Since the output is of type **xs:time**, the output will always have the lexical format **HH:mm:ss**.

Examples

- **altova:parse-time**(xs:string("13:56:24"), "[H]:[m]:[s]") returns 13:56:24
- **altova:parse-time**("13-56-24", "[H]-[m]") returns 13:56:00
- **altova:parse-time**("time=13h56m24s", "time=[H]h[m]m[s]s") returns 13:56:24
- **altova:parse-time**("time=24s56m13h", "time=[s]s[m]m[H]h") returns 13:56:24

[Top]

Age-related functions XP3 XQ3

These functions return the age as calculated (i) between one input argument date and the current date, or (ii) between two input argument dates. The **altova:age** function returns the age in terms of years, the **altova:age-details** function returns the age as a sequence of three integers giving the years, months, and days of the age.

age [altova:]

altova:age(StartDate *as xs:date***)** as **xs:integer** **XP3 XQ3**

Returns an integer that is the age *in years* of some object, counting from a start-date submitted as the argument and ending with the current date (taken from the system clock). If the input argument is a date anything greater than or equal to one year in the future, the return value will be negative.

Examples

If the current date is **2014-01-15**:

- **altova:age**(xs:date("2013-01-15")) returns 1

- **altova:age**(xs:date("2013-01-16")) returns 0
- **altova:age**(xs:date("2015-01-15")) returns -1
- **altova:age**(xs:date("2015-01-14")) returns 0

altova:age(StartDate *as xs:date*, **EndDate** *as xs:date*) as **xs:integer** **XP3** **XQ3**
Returns an integer that is the age *in years* of some object, counting from a start-date that is submitted as the first argument up to an end-date that is the second argument. The return value will be negative if the first argument is one year or more later than the second argument.

Examples

If the current date is **2014-01-15**:

- **altova:age**(xs:date("2000-01-15"), xs:date("2010-01-15")) returns 10
- **altova:age**(xs:date("2000-01-15"), current-date()) returns 14 if the current date is 2014-01-15
- **altova:age**(xs:date("2014-01-15"), xs:date("2010-01-15")) returns -4

age-details [altova:]

altova:age-details(InputDate *as xs:date*) as (**xs:integer**)* **XP3** **XQ3**
Returns three integers that are, respectively, the years, months, and days between the date that is submitted as the argument and the current date (taken from the system clock). The sum of the returned years+months+days together gives the total time difference between the two dates (the input date and the current date). The input date may have a value earlier or later than the current date, but whether the input date is earlier or later is not indicated by the sign of the return values; the return values are always positive.

Examples

If the current date is **2014-01-15**:

- **altova:age-details**(xs:date("2014-01-16")) returns (0 0 1)
- **altova:age-details**(xs:date("2014-01-14")) returns (0 0 1)
- **altova:age-details**(xs:date("2013-01-16")) returns (1 0 1)
- **altova:age-details**(current-date()) returns (0 0 0)

altova:age-details(Date-1 *as xs:date*, **Date-2** *as xs:date*) as (**xs:integer**)* **XP3** **XQ3**
Returns three integers that are, respectively, the years, months, and days between the two argument dates. The sum of the returned years+months+days together gives the total time difference between the two input dates; it does not matter whether the earlier or later of the two dates is submitted as the first argument. The return values do not indicate whether the input date occurs earlier or later than the current date. Return values are always positive.

Examples

- **altova:age-details**(xs:date("2014-01-16"), xs:date("2014-01-15")) returns (0 0 1)
- **altova:age-details**(xs:date("2014-01-15"), xs:date("2014-01-16")) returns (0 0 1)

[Top]

10.1.3 XPath/XQuery Functions: String

The following general-purpose XPath/XQuery extension functions are supported in the current version of your Altova product and can be used in (i) XPath expressions in an XSLT context, or (ii) XQuery expressions in an XQuery document.

Note about naming of functions and language applicability
Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT 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 this namespace. Note that, in future versions of your product, support for a function 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.

XPath functions (used in XPath expressions in XSLT):	XP1 XP2 XP3
XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3
XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3

camel-case [altova:]

altova:camel-case(InputString *as xs:string***)** as **xs:string** XP3 XQ3
Returns the input string **InputString** in CamelCase. The string is analyzed using the regular expression **'\s'** (which is a shortcut for the whitespace character). The first non-whitespace character after a whitespace or sequence of consecutive whitespaces is capitalized. The first character in the output string is capitalized.

Examples

- **altova:camel-case**("max") returns Max
- **altova:camel-case**("max max") returns Max Max
- **altova:camel-case**("file01.xml") returns File01.xml
- **altova:camel-case**("file01.xml file02.xml") returns File01.xml File02.xml
- **altova:camel-case**("file01.xml file02.xml") returns File01.xml File02.xml
- **altova:camel-case**("file01.xml -file02.xml") returns File01.xml -file02.xml

altova:camel-case(InputString *as xs:string*, **SplitChars** *as xs:string*, **IsRegex** *as xs:boolean***)** as **xs:string** XP3 XQ3
Converts the input string **InputString** to camel case by using **SplitChars** to determine the character/s that trigger the next capitalization. **SplitChars** is used as a regular expression when **IsRegex = true()**, or as plain characters when **IsRegex = false()**. The first character in the output string is capitalized.

Examples

- **altova:camel-case**("setname getname", "set|get", true()) returns setName getName
- **altova:camel-case**("altova\documents\testcases", "\", false()) returns Altova\Documents\Testcases

char [altova:]

altova:char(Position *as xs:integer***)** as **xs:string** **XP3** **XQ3**

Returns a string containing the character at the position specified by the `Position` argument, in the string obtained by converting the value of the context item to `xs:string`. The result string will be empty if no character exists at the index submitted by the `Position` argument.

Examples

If the context item is **1234ABCD**:

- **altova:char**(2) returns 2
- **altova:char**(5) returns A
- **altova:char**(9) returns the empty string.
- **altova:char**(-2) returns the empty string.

altova:char(InputString *as xs:string*, **Position** *as xs:integer***)** as **xs:string** **XP3** **XQ3**

Returns a string containing the character at the position specified by the `Position` argument, in the string submitted as the `InputString` argument. The result string will be empty if no character exists at the index submitted by the `Position` argument.

Examples

- **altova:char**("2014-01-15", 5) returns -
- **altova:char**("USA", 1) returns U
- **altova:char**("USA", 10) returns the empty string.
- **altova:char**("USA", -2) returns the empty string.

first-chars [altova:]

altova:first-chars(X-Number *as xs:integer***)** as **xs:string** **XP3** **XQ3**

Returns a string containing the first `X-Number` of characters of the string obtained by converting the value of the context item to `xs:string`.

Examples

If the context item is **1234ABCD**:

- **altova:first-chars**(2) returns 12
- **altova:first-chars**(5) returns 1234A
- **altova:first-chars**(9) returns 1234ABCD

altova:first-chars(InputString *as xs:string*, **X-Number** *as xs:integer***)** as **xs:string** **XP3** **XQ3**

Returns a string containing the first `X-Number` of characters of the string submitted as the `InputString` argument.

Examples

- **altova:first-chars**("2014-01-15", 5) returns 2014-
- **altova:first-chars**("USA", 1) returns U

last-chars [altova:]

altova:last-chars(X-Number *as xs:integer***)** as **xs:string** **XP3** **XQ3**

Returns a string containing the last `X-Number` of characters of the string obtained by converting the value of the context item to `xs:string`.

Examples

If the context item is **1234ABCD**:

- `altova:last-chars(2)` returns `CD`
- `altova:last-chars(5)` returns `4ABCD`
- `altova:last-chars(9)` returns `1234ABCD`

altova:last-chars(InputString *as xs:string*, **X-Number** *as xs:integer*) as **xs:string** **XP3 XQ3**

Returns a string containing the last `X-Number` of characters of the string submitted as the `InputString` argument.

Examples

- `altova:last-chars("2014-01-15", 5)` returns `01-15`
- `altova:last-chars("USA", 10)` returns `USA`

pad-string-left [altova:]

altova:pad-string-left(StringToPad *as xs:string*, **Repeats** *as xs:integer*, **PadCharacter** *as xs:string*) as **xs:string** **XP3 XQ3**

The `PadCharacter` argument is a single character that is padded to the left of the string submitted as the `StringToPad` argument. The `Repeats` argument gives the number of times the pad-character is to be repeated at the left of `StringToPad`.

Examples

- `altova:pad-string-left('AP', 1, 'Z')` returns `'ZAP'`
- `altova:pad-string-left('AP', 3, 'Z')` returns `'ZZZAP'`
- `altova:pad-string-left('AP', 0, 'Z')` returns `'AP'`
- `altova:pad-string-left('AP', 3, 'YZ')` returns a pad-character-too-long error

pad-string-right [altova:]

altova:pad-string-right(StringToPad *as xs:string*, **Repeats** *as xs:integer*, **PadCharacter** *as xs:string*) as **xs:string** **XP3 XQ3**

The `PadCharacter` argument is a single character that is padded to the right of the string submitted as the `StringToPad` argument. The `Repeats` argument gives the number of times the pad-character is to be repeated at the right of `StringToPad`.

Examples

- `altova:pad-string-right('AP', 1, 'Z')` returns `'APZ'`
- `altova:pad-string-right('AP', 3, 'Z')` returns `'APZZZ'`
- `altova:pad-string-right('AP', 0, 'Z')` returns `'AP'`
- `altova:pad-string-right('AP', 3, 'YZ')` returns a pad-character-too-long error

repeat-string [altova:]

altova:repeat-string(InputString *as xs:string*, **Repeats** *as xs:integer*) as **xs:string** **XP2 XQ1 XP3 XQ3**

Generates a string that is composed of the first `InputString` argument repeated `Repeats` number of times.

Examples

- `altova:repeat-string("Altova #", 3)` returns `"Altova #Altova #Altova #"`

substring-after-last [altova:]

altova:substring-after-last(MainString *as xs:string*, **CheckString** *as xs:string*) as **xs:string** **XP3** **XQ3**

If `CheckString` is found in `MainString`, then the substring that occurs after `CheckString` in `MainString` is returned. If `CheckString` is not found in `MainString`, then the empty string is returned. If `CheckString` is an empty string, then `MainString` is returned in its entirety. If there is more than one occurrence of `CheckString` in `MainString`, then the substring after the last occurrence of `CheckString` is returned.

Examples

- **altova:substring-after-last**('ABCDEFGH', 'B') returns 'CDEFGH'
- **altova:substring-after-last**('ABCDEFGH', 'BC') returns 'DEFGH'
- **altova:substring-after-last**('ABCDEFGH', 'BD') returns ''
- **altova:substring-after-last**('ABCDEFGH', 'Z') returns ''
- **altova:substring-after-last**('ABCDEFGH', '') returns 'ABCDEFGH'
- **altova:substring-after-last**('ABCD-ABCD', 'B') returns 'CD'
- **altova:substring-after-last**('ABCD-ABCD-ABCD', 'BCD') returns ''

substring-before-last [altova:]

altova:substring-before-last(MainString *as xs:string*, **CheckString** *as xs:string*) as **xs:string** **XP3** **XQ3**

If `CheckString` is found in `MainString`, then the substring that occurs before `CheckString` in `MainString` is returned. If `CheckString` is not found in `MainString`, or if `CheckString` is an empty string, then the empty string is returned. If there is more than one occurrence of `CheckString` in `MainString`, then the substring before the last occurrence of `CheckString` is returned.

Examples

- **altova:substring-before-last**('ABCDEFGH', 'B') returns 'A'
- **altova:substring-before-last**('ABCDEFGH', 'BC') returns 'A'
- **altova:substring-before-last**('ABCDEFGH', 'BD') returns ''
- **altova:substring-before-last**('ABCDEFGH', 'Z') returns ''
- **altova:substring-before-last**('ABCDEFGH', '') returns ''
- **altova:substring-before-last**('ABCD-ABCD', 'B') returns 'ABCD-A'
- **altova:substring-before-last**('ABCD-ABCD-ABCD', 'ABCD') returns 'ABCD-ABCD-'

substring-pos [altova:]

altova:substring-pos(StringToCheck *as xs:string*, **StringToFind** *as xs:string*) as **xs:integer** **XP3** **XQ3**

Returns the character position of the first occurrence of `StringToFind` in the string `StringToCheck`. The character position is returned as an integer. The first character of `StringToCheck` has the position `1`. If `StringToFind` does not occur within `StringToCheck`, the integer `0` is returned. To check for the second or a later occurrence of `StringToCheck`, use the next signature of this function.

Examples

- **altova:substring-pos**('Altova', 'to') returns 3
- **altova:substring-pos**('Altova', 'tov') returns 3
- **altova:substring-pos**('Altova', 'tv') returns 0
- **altova:substring-pos**('AltovaAltova', 'to') returns 3

altova:substring-pos(StringToCheck *as xs:string*, **StringToFind** *as xs:string*, **Integer** *as xs:integer*) as **xs:integer** XP3 XQ3

Returns the character position of `StringToFind` in the string, `StringToCheck`. The search for `StringToFind` starts from the character position given by the `Integer` argument; the character substring before this position is not searched. The returned integer, however, is the position of the found string within the *entire* string, `StringToCheck`. This signature is useful for finding the second or a later position of a string that occurs multiple times with the `StringToCheck`. If `StringToFind` does not occur within `StringToCheck`, the integer `0` is returned.

Examples

- `altova:substring-pos('Altova', 'to', 1)` returns `3`
- `altova:substring-pos('Altova', 'to', 3)` returns `3`
- `altova:substring-pos('Altova', 'to', 4)` returns `0`
- `altova:substring-pos('Altova-Altova', 'to', 0)` returns `3`
- `altova:substring-pos('Altova-Altova', 'to', 4)` returns `10`

trim-string [altova:]

altova:trim-string(InputString *as xs:string*) as **xs:string** XP3 XQ3

This function takes an `xs:string` argument, removes any leading and trailing whitespace, and returns a "trimmed" `xs:string`.

Examples

- `altova:trim-string("  Hello World  ")` returns `"Hello World"`
- `altova:trim-string("Hello World  ")` returns `"Hello World"`
- `altova:trim-string("  Hello World")` returns `"Hello World"`
- `altova:trim-string("Hello World")` returns `"Hello World"`
- `altova:trim-string("Hello  World")` returns `"Hello  World"`

trim-string-left [altova:]

altova:trim-string-left(InputString *as xs:string*) as **xs:string** XP3 XQ3

This function takes an `xs:string` argument, removes any leading whitespace, and returns a left-trimmed `xs:string`.

Examples

- `altova:trim-string-left("  Hello World  ")` returns `"Hello World  "`
- `altova:trim-string-left("Hello World  ")` returns `"Hello World  "`
- `altova:trim-string-left("  Hello World")` returns `"Hello World"`
- `altova:trim-string-left("Hello World")` returns `"Hello World"`
- `altova:trim-string-left("Hello  World")` returns `"Hello  World"`

trim-string-right [altova:]

altova:trim-string-right(InputString *as xs:string*) as **xs:string** XP3 XQ3

This function takes an `xs:string` argument, removes any trailing whitespace, and returns a right-trimmed `xs:string`.

Examples

- `altova:trim-string-right("  Hello World  ")` returns `"  Hello World"`
- `altova:trim-string-right("Hello World  ")` returns `"Hello World"`
- `altova:trim-string-right("  Hello World")` returns `"  Hello World"`

- `altova:trim-string-right("Hello World")` returns `"Hello World"`
- `altova:trim-string-right("Hello  World")` returns `"Hello  World"`

10.1.4 XPath/XQuery Functions: Miscellaneous

The following general-purpose XPath/XQuery extension functions are supported in the current version of your Altova product and can be used in (i) XPath expressions in an XSLT context, or (ii) XQuery expressions in an XQuery document.

Note about naming of functions and language applicability
Altova extension functions can be used in XPath/XQuery expressions. They provide additional functionality to the functionality that is available in the standard library of XPath, XQuery, and XSLT 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 this namespace. Note that, in future versions of your product, support for a function 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.

XPath functions (used in XPath expressions in XSLT):	XP1 XP2 XP3
XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3
XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3

Auto-numbering functions

generate-auto-number [altova:]

altova:generate-auto-number(ID *as xs:string*, **StartsWith** *as xs:double*, **Increment** *as xs:double*, **ResetOnChange** *as xs:string***)** as xs:integer XP1 XP2 XQ1 XP3 XQ3

Generates a number each time the function is called. The first number, which is generated the first time the function is called, is specified by the `StartsWith` argument. Each subsequent call to the function generates a new number, this number being incremented over the previously generated number by the value specified in the `Increment` argument. In effect, the `altova:generate-auto-number` function creates a counter having a name specified by the `ID` argument, with this counter being incremented each time the function is called. If the value of the `ResetOnChange` argument changes from that of the previous function call, then the value of the number to be generated is reset to the `StartsWith` value. Auto-numbering can also be reset by using the `altova:reset-auto-number` function.

Examples

- **altova:generate-auto-number**`("ChapterNumber", 1, 1, "SomeString")` will return one number each time the function is called, starting with `1`, and incrementing by `1` with each call to the function. As long as the fourth argument remains `"SomeString"` in each subsequent call, the incrementing will continue. When the value of the fourth argument changes, the counter (called `ChapterNumber`) will reset to 1. The value of `ChapterNumber` can also be reset by a call to the `altova:reset-auto-number` function, like this: `altova:reset-auto-number("ChapterNumber")`.

reset-auto-number [altova:]

altova:reset-auto-number(ID *as xs:string***)** XP1 XP2 XQ1 XP3 XQ3

This function resets the number of the auto-numbering counter named in the `ID` argument.

The number is reset to the number specified by the `StartsWith` argument of the `altova:generate-auto-number` function that created the counter named in the `ID` argument.

Examples

- `altova:reset-auto-number("ChapterNumber")` resets the number of the auto-numbering counter named `ChapterNumber` that was created by the `altova:generate-auto-number` function. The number is reset to the value of the `StartsWith` argument of the `altova:generate-auto-number` function that created `ChapterNumber`.

[Top]

Numeric functions

hex-string-to-integer [altova:]

`altova:hex-string-to-integer(HexString as xs:string) as xs:integer` XP3 XQ3

Takes a string argument that is the Base-16 equivalent of an integer in the decimal system (Base-10), and returns the decimal integer.

Examples

- `altova:hex-string-to-integer('1')` returns `1`
- `altova:hex-string-to-integer('9')` returns `9`
- `altova:hex-string-to-integer('A')` returns `10`
- `altova:hex-string-to-integer('B')` returns `11`
- `altova:hex-string-to-integer('F')` returns `15`
- `altova:hex-string-to-integer('G')` returns an error
- `altova:hex-string-to-integer('10')` returns `16`
- `altova:hex-string-to-integer('01')` returns `1`
- `altova:hex-string-to-integer('20')` returns `32`
- `altova:hex-string-to-integer('21')` returns `33`
- `altova:hex-string-to-integer('5A')` returns `90`
- `altova:hex-string-to-integer('USA')` returns an error

integer-to-hex-string [altova:]

`altova:integer-to-hex-string(Integer as xs:integer) as xs:string` XP3 XQ3

Takes an integer argument and returns its Base-16 equivalent as a string.

Examples

- `altova:integer-to-hex-string(1)` returns `'1'`
- `altova:integer-to-hex-string(9)` returns `'9'`
- `altova:integer-to-hex-string(10)` returns `'A'`
- `altova:integer-to-hex-string(11)` returns `'B'`
- `altova:integer-to-hex-string(15)` returns `'F'`
- `altova:integer-to-hex-string(16)` returns `'10'`
- `altova:integer-to-hex-string(32)` returns `'20'`
- `altova:integer-to-hex-string(33)` returns `'21'`
- `altova:integer-to-hex-string(90)` returns `'5A'`

[Top]

Sequence functions

attributes [altova:]

altova:attributes(AttributeName *as xs:string*) as **attribute()*** **XP3 XQ3**

Returns all attributes that have a local name which is the same as the name supplied in the input argument, `AttributeName`. The search is case-sensitive and conducted along the `attribute::` axis.

Examples

- **altova:attributes**("MyAttribute") returns `MyAttribute()*`

altova:attributes(AttributeName *as xs:string*, **SearchOptions** *as xs:string*) as **attribute()*** **XP3 XQ3**

Returns all attributes that have a local name which is the same as the name supplied in the input argument, `AttributeName`. The search is case-sensitive and conducted along the `attribute::` axis. The second argument is a string containing option flags. Available flags are:

r = switches to a regular-expression search; `AttributeName` must then be a regular-expression search string;
i = switches to a case-insensitive search;
p = includes the namespace prefix in the search; `AttributeName` should then contain the namespace prefix, for example: `altova:MyAttribute`.
The flags can be written in any order. Invalid flags will generate errors. One or more flags can be omitted. The empty string is allowed, and will produce the same effect as the function having only one argument (*previous signature*). However, an empty sequence is not allowed.

Examples

- **altova:attributes**("MyAttribute", "rip") returns `MyAttribute()*`
- **altova:attributes**("MyAttribute", "pri") returns `MyAttribute()*`
- **altova:attributes**("MyAttribute", "") returns `MyAttribute()*`
- **altova:attributes**("MyAttribute", "Rip") returns an unrecognized-flag error.
- **altova:attributes**("MyAttribute",) returns a missing-second-argument error.

elements [altova:]

altova:elements(ElementName *as xs:string*) as **element()*** **XP3 XQ3**

Returns all elements that have a local name which is the same as the name supplied in the input argument, `ElementName`. The search is case-sensitive and conducted along the `child::` axis.

Examples

- **altova:elements**("MyElement") returns `MyElement()*`

altova:elements(ElementName *as xs:string*, **SearchOptions** *as xs:string*) as **element()*** **XP3 XQ3**

Returns all elements that have a local name which is the same as the name supplied in the input argument, `ElementName`. The search is case-sensitive and conducted along the `child::` axis. The second argument is a string containing option flags. Available flags are:

r = switches to a regular-expression search; `ElementName` must then be a regular-expression search string;

i = switches to a case-insensitive search;
p = includes the namespace prefix in the search; ElementName should then contain the namespace prefix, for example: altova:MyElement.
The flags can be written in any order. Invalid flags will generate errors. One or more flags can be omitted. The empty string is allowed, and will produce the same effect as the function having only one argument (*previous signature*). However, an empty sequence is not allowed.

Examples

- **altova:elements**("MyElement", "rip") returns MyElement()*
- **altova:elements**("MyElement", "pri") returns MyElement()*
- **altova:elements**("MyElement", "") returns MyElement()*
- **altova:elements**("MyElement", "Rip") returns an unrecognized-flag error.
- **altova:elements**("MyElement",) returns a missing-second-argument error.

find-first [altova:]

altova:find-first((Sequence *as item()**)**, **(Condition(Sequence-Item** *as xs:boolean***))** as **item()?** **XP3** **XQ3**

This function takes two arguments. The first argument is a sequence of one or more items of any datatype. The second argument, Condition, is a reference to an XPath function that takes one argument (has an arity of **1**) and returns a boolean. Each item of **Sequence** is submitted, in turn, to the function referenced in Condition. (*Remember:* This function takes a single argument.) The first **Sequence** item that causes the function in **Condition** to evaluate to **true()** is returned as the result of **altova:find-first**, and the iteration stops.

Examples

- **altova:find-first**(5 to 10, function($a) {$a mod 2 = 0}) returns xs:integer 6

  The **Condition** argument references the XPath 3.0 inline function, **function()**, which declares an inline function named **$a** and then defines it. Each item in the **Sequence** argument of **altova:find-first** is passed, in turn, to **$a** as its input value. The input value is tested on the condition in the function definition ($a mod 2 = 0). The first input value to satisfy this condition is returned as the result of **altova:find-first** (in this case **6**).

- **altova:find-first**((1 to 10), (function($a) {$a+3=7})) returns xs:integer 4

Further examples

If the file **C:\Temp\Customers.xml** exists:

- **altova:find-first**(("C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1)) returns xs:string C:\Temp\Customers.xml

If the file **C:\Temp\Customers.xml** does not exist, and **http://www.altova.com/index.html** exists:

- **altova:find-first**(("C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1)) returns xs:string http://www.altova.com/index.html

If the file **C:\Temp\Customers.xml** does not exist, and **http://www.altova.com/**

index.html also does not exist:

- **altova:find-first**(("C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1)) returns no result

Notes about the examples given above

- The XPath 3.0 function, doc-available, takes a single string argument, which is used as a URI, and returns true if a document node is found at the submitted URI. (The document at the submitted URI must therefore be an XML document.)
- The doc-available function can be used for **Condition**, the second argument of altova:find-first, because it takes only one argument (arity=1), because it takes an item() as input (a string which is used as a URI), and returns a boolean value.
- Notice that the doc-available function is only referenced, not called. The #1 suffix that is attached to it indicates a function with an arity of 1. In its entirety doc-available#1 simply means: *Use the doc-availabe() function that has arity=1, passing to it as its single argument, in turn, each of the items in the first sequence*. As a result, each of the two strings will be passed to **doc-available()**, which uses the string as a URI and tests whether a document node exists at the URI. If one does, the **doc-available()** evaluates to true() and that string is returned as the result of the **altova:find-first** function. *Note about the doc-available() function: Relative paths are resolved relative to the the current base URI, which is by default the URI of the XML document from which the function is loaded.*

find-first-combination [altova:]

altova:find-first-combination((Seq-01 *as item()**), **(Seq-02** *as item()**), **(Condition(Seq-01-Item, Seq-02-Item** *as xs:boolean*)) as **item()*** **XP3** **XQ3**

This function takes three arguments:

- The first two arguments, **Seq-01** and **Seq-02**, are sequences of one or more items of any datatype.
- The third argument, **Condition**, is a reference to an XPath function that takes two arguments (has an arity of 2) and returns a boolean.

The items of **Seq-01** and **Seq-02** are passed in ordered pairs (one item from each sequence making up a pair) as the arguments of the function in **Condition**. The pairs are ordered as follows.

```
If   Seq-01 = X1, X2, X3 ... Xn
And  Seq-02 = Y1, Y2, Y3 ... Yn
Then (X1 Y1), (X1 Y2), (X1 Y3) ... (X1 Yn), (X2 Y1), (X2 Y2) ... (Xn Yn)
```

The first ordered pair that causes the **Condition** function to evaluate to **true()** is returned as the result of **altova:find-first-combination**. Note that: (i) If the **Condition** function iterates through the submitted argument pairs and does not once evaluate to **true()**, then **altova:find-first-combination** returns *No results*; (ii) The result of **altova:find-first-combination** will always be a pair of items (of any datatype) or no item at all.

Examples

- **altova:find-first-pair**(11 to 20, 21 to 30, function($a, $b) {$a+$b = 32}) returns the sequence of xs:integers (11, 21)
- **altova:find-first-pair**(11 to 20, 21 to 30, function($a, $b) {$a+$b = 33}) returns the sequence of xs:integers (11, 22)
- **altova:find-first-pair**(11 to 20, 21 to 30, function($a, $b) {$a+$b =

34})` returns the sequence of `xs:integers (11, 23)`

find-first-pair [altova:]

altova:find-first-pair((Seq-01 *as item()**)**, **(Seq-02** *as item()**)**,
(Condition(Seq-01-Item, Seq-02-Item *as xs:boolean*)) as **item()*** **XP3** **XQ3**

This function takes three arguments:

- The first two arguments, **Seq-01** and **Seq-02**, are sequences of one or more items of any datatype.
- The third argument, **Condition**, is a reference to an XPath function that takes two arguments (has an arity of 2) and returns a boolean.

The items of **Seq-01** and **Seq-02** are passed in ordered pairs as the arguments of the function in **Condition**. The pairs are ordered as follows.

```
If   Seq-01 = X1, X2, X3 ... Xn
And  Seq-02 = Y1, Y2, Y3 ... Yn
Then (X1 Y1), (X2 Y2), (X3 Y3) ... (Xn Yn)
```

The first ordered pair that causes the **Condition** function to evaluate to **true()** is returned as the result of **altova:find-first-pair**. Note that: (i) If the **Condition** function iterates through the submitted argument pairs and does not once evaluate to **true()**, then **altova:find-first-pair** returns *No results*; (ii) The result of **altova:find-first-pair** will always be a pair of items (of any datatype) or no item at all.

Examples

- **altova:find-first-pair**`(11 to 20, 21 to 30, function($a, $b) {$a+$b = 32})` returns the sequence of `xs:integers (11, 21)`
- **altova:find-first-pair**`(11 to 20, 21 to 30, function($a, $b) {$a+$b = 33})` returns *No results*

Notice from the two examples above that the ordering of the pairs is: `(11, 21) (12, 22) (13, 23)...(20, 30)`. This is why the second example returns *No results* (because no ordered pair gives a sum of `33`).

find-first-pair-pos [altova:]

altova:find-first-pair-pos((Seq-01 *as item()**)**, **(Seq-02** *as item()**)**,
(Condition(Seq-01-Item, Seq-02-Item *as xs:boolean*)) as **xs:integer** **XP3** **XQ3**

This function takes three arguments:

- The first two arguments, **Seq-01** and **Seq-02**, are sequences of one or more items of any datatype.
- The third argument, **Condition**, is a reference to an XPath function that takes two arguments (has an arity of 2) and returns a boolean.

The items of **Seq-01** and **Seq-02** are passed in ordered pairs as the arguments of the function in **Condition**. The pairs are ordered as follows.

```
If   Seq-01 = X1, X2, X3 ... Xn
And  Seq-02 = Y1, Y2, Y3 ... Yn
Then (X1 Y1), (X2 Y2), (X3 Y3) ... (Xn Yn)
```


The index position of the first ordered pair that causes the **Condition** function to evaluate to **true()** is returned as the result of **altova:find-first-pair-pos**. Note that if the **Condition** function iterates through the submitted argument pairs and does not once evaluate to **true()**, then **altova:find-first-pair-pos** returns *No results*.

Examples

- **altova:find-first-pair**(11 to 20, 21 to 30, function($a, $b) {$a+$b = 32}) returns 1
- **altova:find-first-pair**(11 to 20, 21 to 30, function($a, $b) {$a+$b = 33}) returns *No results*

Notice from the two examples above that the ordering of the pairs is: (11, 21) (12, 22) (13, 23)...(20, 30). In the first example, the first pair causes the **Condition** function to evaluate to **true()**, and so its index position in the sequence, **1**, is returned. The second example returns *No results* because no pair gives a sum of 33.

find-first-pos [altova:]

altova:find-first-pos((Sequence *as item()**)**,* **(Condition(** **Sequence-Item** *as xs:boolean)) as* **xs:integer** **XP3** **XQ3**

This function takes two arguments. The first argument is a sequence of one or more items of any datatype. The second argument, Condition, is a reference to an XPath function that takes one argument (has an arity of **1**) and returns a boolean. Each item of **Sequence** is submitted, in turn, to the function referenced in Condition. (*Remember:* This function takes a single argument.) The first **Sequence** item that causes the function in **Condition** to evaluate to **true()** has its index position in **Sequence** returned as the result of **altova:find-first-pos**, and the iteration stops.

Examples

- **altova:find-first-pos**(5 to 10, function($a) {$a mod 2 = 0}) returns xs:integer 2

The **Condition** argument references the XPath 3.0 inline function, **function()**, which declares an inline function named **$a** and then defines it. Each item in the **Sequence** argument of **altova:find-first-pos** is passed, in turn, to **$a** as its input value. The input value is tested on the condition in the function definition ($a mod 2 = 0). The index position in the sequence of the first input value to satisfy this condition is returned as the result of **altova:find-first-pos** (in this case **2**, since **6**, the first value (in the sequence) to satisfy the condition, is at index position **2** in the sequence).

- **altova:find-first-pos**((2 to 10), (function($a) {$a+3=7})) returns xs:integer 3

Further examples

If the file **C:\Temp\Customers.xml** exists:

- **altova:find-first-pos**(("C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1)) returns 1

If the file **C:\Temp\Customers.xml** does not exist, and **http://www.altova.com/index.html** exists:

- **altova:find-first-pos**(("C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1)) returns 2

If the file **C:\Temp\Customers.xml** does not exist, and **http://www.altova.com/index.html** also does not exist:

- **altova:find-first-pos**(("C:\Temp\Customers.xml", "http://www.altova.com/index.html"), (doc-available#1)) returns no result

Notes about the examples given above

- The XPath 3.0 function, doc-available, takes a single string argument, which is used as a URI, and returns true if a document node is found at the submitted URI. (The document at the submitted URI must therefore be an XML document.)
- The doc-available function can be used for **Condition**, the second argument of altova:find-first-pos, because it takes only one argument (arity=1), because it takes an item() as input (a string which is used as a URI), and returns a boolean value.
- Notice that the doc-available function is only referenced, not called. The #1 suffix that is attached to it indicates a function with an arity of 1. In its entirety doc-available#1 simply means: *Use the doc-availabe() function that has arity=1, passing to it as its single argument, in turn, each of the items in the first sequence*. As a result, each of the two strings will be passed to **doc-available()**, which uses the string as a URI and tests whether a document node exists at the URI. If one does, the **doc-available()** function evaluates to true() and the index position of that string in the sequence is returned as the result of the **altova:find-first-pos** function. *Note about the doc-available() function: Relative paths are resolved relative to the the current base URI, which is by default the URI of the XML document from which the function is loaded.*

substitute-empty [altova:]

altova:substitute-empty(FirstSequence *as item()**, **SecondSequence** *as item()***)** as **item()*** **XP3** **XQ3**

If FirstSequence is empty, returns SecondSequence. If FirstSequence is not empty, returns FirstSequence.

Examples

- **altova:substitute-empty**((1,2,3), (4,5,6)) returns (1,2,3)
- **altova:substitute-empty**((), (4,5,6)) returns (4,5,6)

[Top]

URI functions

get-temp-folder [altova:]

altova:get-temp-folder() as xs:string **XP2** **XQ1** **XP3** **XQ3**

This function takes no argument. It returns the path to the temporary folder of the current user.

Examples

- **altova:get-temp-folder**() would return, on a Windows machine, something like C:\Users\<UserName>\AppData\Local\Temp\ as an xs:string.

[Top]

10.1.5 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-*N*y-row extension item

where

- x is the value of the X-Axis column of the chart data row
- yN are the values of the Y-Axis columns

The function returns a chart-data-x-Ny-row extension item, which contains the data for the X-

Axis column and Y-Axis columns of a single series.

---

altova:create-chart-data-series-from-rows(`$series-names as xs:string*, $row*) as chart-data-series extension item`

where

- `$series-name` is the name of the series to be created
- `$row` is the chart-data-x-Ny-row extension item that is to be created as a series

The function returns a chart-data-series extension item, which contains the data for the X-Axis and Y-Axes of the series.

---

altova:create-chart-layer(`$chart-config, $chart-data-series*) as chart-layer extension item`

where

- `$chart-config` is the chart-config extension item obtained with the `altova:create-chart-config` function or or via the `altova:create-chart-config-from-xml` function
- `$chart-data-series` is the chart-data-series extension item obtained with the `altova:create-chart-data-series` function or `altova:create-chart-data-series-from-rows` function

The function returns a chart-layer extension item, which contains chart-layer data.

---

altova:create-multi-layer-chart(`$chart-config, $chart-data-series*, $chart-layer*)`

where

- `$chart-config` is the chart-config extension item obtained with the `altova:create-chart-config` function or or via the `altova:create-chart-config-from-xml` function
- `$chart-data-series` is the chart-data-series extension item obtained with the `altova:create-chart-data-series` function or `altova:create-chart-data-series-from-rows` function
- `$chart-layer` is the chart-layer extension item obtained with the `altova:create-chart-layer` function

The function returns a multi-layer-chart item.

---

altova:create-multi-layer-chart($chart-config, $chart-data-series*, $chart-layer*, xs:boolean $mergecategoryvalues)

where

- $chart-config is the chart-config extension item obtained with the altova:create-chart-config function or or via the altova:create-chart-config-from-xml function
- $chart-data-series is the chart-data-series extension item obtained with the altova:create-chart-data-series function or altova:create-chart-data-series-from-rows function
- $chart-layer is the chart-layer extension item obtained with the altova:create-chart-layer function

The function returns a multi-layer-chart item.

Chart Data XML Structure

Given below is the XML structure of chart data, how it might appear for the Altova extension functions for charts. This affects the appearance of the specific chart. Not all elements are used for all chart kinds, e.g. the <Pie> element is ignored for bar charts.

Note: Chart functions are supported only in the **Enterprise and 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>
```

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>
```

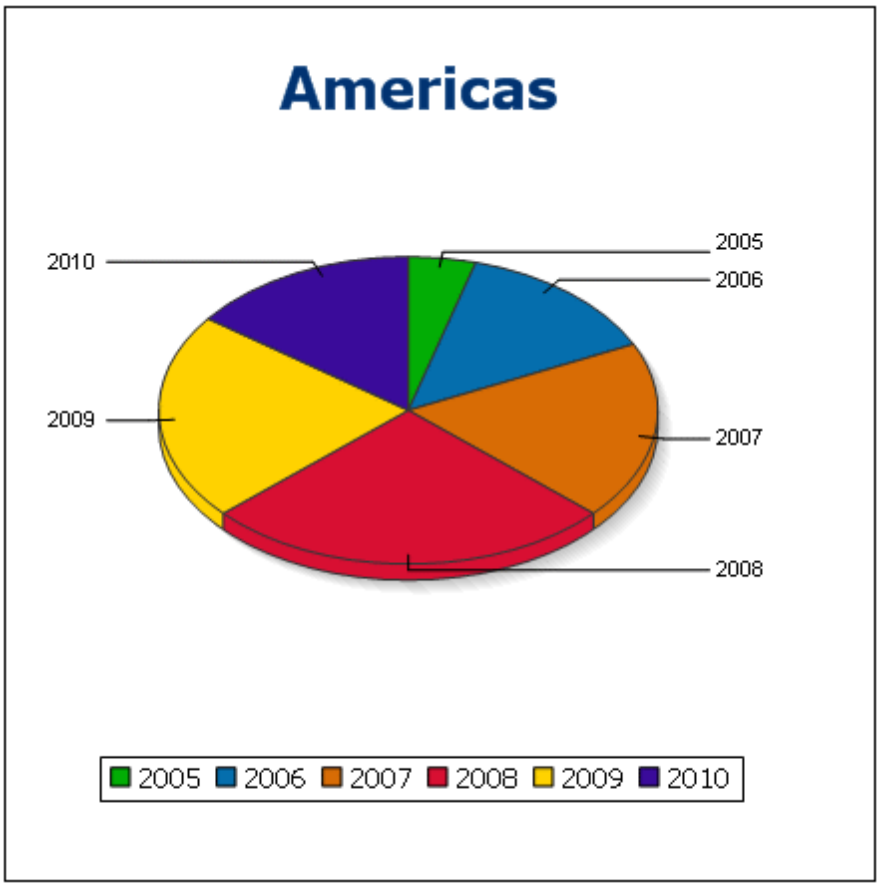
XML document

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.



10.1.6 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\Common2015\jar`.

The Java libraries used are in sub-folders of the folder `<ProgramFilesFolder>\Altova\Common2015\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:generateBarcodePngAsHexString($barcodeObject)) ))"/>
      </xsl:result-document>
   </xsl:template>
</xsl:stylesheet>
```


10.2 Miscellaneous 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, as well as MSXSL scripts for XSLT. They also support XBRL functions for XSLT. 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:

- 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.

10.2.1 Java Extension Functions

A Java extension function can be used within an XPath or XQuery expression to invoke a Java constructor or call a Java method (static or instance).

A field in a Java class is considered to be a method without any argument. A field can be static or instance. How to access fields is described in the respective sub-sections, static and instance.

This section is organized into the following sub-sections:

- Java: Constructors
- Java: Static Methods and Static Fields
- Java: Instance Methods and Instance Fields
- Datatypes: XPath/XQuery to Java
- Datatypes: Java to XPath/XQuery

Form of the extension function
The extension function in the XPath/XQuery expression must have the form `prefix:fname()`.

- The `prefix:` part identifies the extension function as a Java function. It does so by associating the extension function with an in-scope namespace declaration, the URI of which must begin with `java:` (*see below for examples*). The namespace declaration should identify a Java class, for example: `xmlns:myns="java:java.lang.Math"`. However, it could also simply be: `xmlns:myns="java"` (without a colon), with the identification of the Java class being left to the `fname()` part of the extension function.
- The `fname()` part identifies the Java method being called, and supplies the arguments for the method (*see below for examples*). However, if the namespace URI identified by the `prefix:` part does not identify a Java class (*see preceding point*), then the Java class should be identified in the `fname()` part, before the class and separated from the class by a period (*see the second XSLT example below*).

Note: The class being called must be on the classpath of the machine.

XSLT example
Here are two examples of how a static method can be called. In the first example, the class name (`java.lang.Math`) is included in the namespace URI and, therefore, must not be in the `fname()` part. In the second example, the `prefix:` part supplies the prefix `java:` while the `fname()` part identifies the class as well as the method.

```
<xsl:value-of xmlns:jMath="java:java.lang.Math"
              select="jMath:cos(3.14)" />

<xsl:value-of xmlns:jmath="java"
              select="jmath:java.lang.Math.cos(3.14)" />
```

The method named in the extension function (`cos()` in the example above) must match the name of a public static method in the named Java class (`java.lang.Math` in the example above).

XQuery example
Here is an XQuery example similar to the XSLT example above:

```
<cosine xmlns:jMath="java:java.lang.Math">
   {jMath:cos(3.14)}
</cosine>
```

User-defined Java classes
If you have created your own Java classes, methods in these classes are called differently according to: (i) whether the classes are accessed via a JAR file or a class file, and (ii) whether these files (JAR or class) are located in the current directory (the same directory as the XSLT or XQuery document) or not. How to locate these files is described in the sections User-Defined Class Files and User-Defined Jar Files. Note that paths to class files not in the current directory and to all JAR files must be specified.

User-Defined Class Files

If access is via a class file, then there are two possibilities:

- The class file is in a package. The XSLT or XQuery file is in the same folder as the Java package. (*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.

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.

Java: Constructors

An extension function can be used to call a Java constructor. All constructors are called with the pseudo-function `new()`.

If the result of a Java constructor call can be implicitly converted to XPath/XQuery datatypes, then the Java extension function will return a sequence that is an XPath/XQuery datatype. If the result of a Java constructor call cannot be converted to a suitable XPath/XQuery datatype, then the constructor creates a wrapped Java object with a type that is the name of the class returning that Java object. For example, if a constructor for the class `java.util.Date` is called (`java.util.Date.new()`), then an object having a type `java.util.Date` is returned. The lexical format of the returned object may not match the lexical format of an XPath datatype and the value would therefore need to be converted to the lexical format of the required XPath datatype and then to the required XPath datatype.

There are two things that can be done with a Java object created by a constructor:

- It can be assigned to a variable:
  ```
  <xsl:variable name="currentdate" select="date:new()"
  xmlns:date="java:java.util.Date" />
  ```
- It can be passed to an extension function (*see Instance Method and Instance Fields*):
  ```
  <xsl:value-of select="date:toString(date:new())"
  xmlns:date="java:java.util.Date" />
  ```

Java: Static Methods and Static Fields

A static method is called directly by its Java name and by supplying the arguments for the method. Static fields (methods that take no arguments), such as the constant-value fields `E` and `PI`, are accessed without specifying any argument.

XSLT examples
Here are some examples of how static methods and fields can be called:

```
<xsl:value-of xmlns:jMath="java:java.lang.Math"
              select="jMath:cos(3.14)" />
```

```
<xsl:value-of xmlns:jMath="java:java.lang.Math"
              select="jMath:cos( jMath:PI() )" />

<xsl:value-of xmlns:jMath="java:java.lang.Math"
              select="jMath:E() * jMath:cos(3.14)" />
```

Notice that the extension functions above have the form `prefix:fname()`. The prefix in all three cases is `jMath:`, which is associated with the namespace URI `java:java.lang.Math`. (The namespace URI must begin with `java:`. In the examples above it is extended to contain the class name (`java.lang.Math`).) The `fname()` part of the extension functions must match the name of a public class (e.g. `java.lang.Math`) followed by the name of a public static method with its argument/s (such as `cos(3.14)`) or a public static field (such as `PI()`).

In the examples above, the class name has been included in the namespace URI. If it were not contained in the namespace URI, then it would have to be included in the `fname()` part of the extension function. For example:

```
<xsl:value-of xmlns:java="java:"
              select="java:java.lang.Math.cos(3.14)" />
```

XQuery example
A similar example in XQuery would be:

```
<cosine xmlns:jMath="java:java.lang.Math">
   {jMath:cos(3.14)}
</cosine>
```

Java: Instance Methods and Instance Fields

An instance method has a Java object passed to it as the first argument of the method call. Such a Java object typically would be created by using an extension function (for example a constructor call) or a stylesheet parameter/variable. An XSLT example of this kind would be:

```
<xsl:stylesheet version="1.0" exclude-result-prefixes="date"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:date="java:java.util.Date"
  xmlns:jlang="java:java.lang">
  <xsl:param name="CurrentDate" select="date:new()"/>
  <xsl:template match="/">
      <enrollment institution-id="Altova School"
                date="{date:toString($CurrentDate)}"
                type="
{jlang:Object.toString(jlang:Object.getClass( date:new() ))}">
      </enrollment>
  </xsl:template>
</xsl:stylesheet>
```

In the example above, the value of the node `enrollment/@type` is created as follows:

1. An object is created with a constructor for the class `java.util.Date` (with the `date:new()` constructor).
2. This Java object is passed as the argument of the `jlang.Object.getClass` method.

3. The object obtained by the `getClass` method is passed as the argument to the `jlang.Object.toString` method.

The result (the value of `@type`) will be a string having the value: `java.util.Date`.

An instance field is theoretically different from an instance method in that it is not a Java object per se that is passed as an argument to the instance field. Instead, a parameter or variable is passed as the argument. However, the parameter/variable may itself contain the value returned by a Java object. For example, the parameter `CurrentDate` takes the value returned by a constructor for the class `java.util.Date`. This value is then passed as an argument to the instance method `date:toString` in order to supply the value of `/enrollment/@date`.

Datatypes: XPath/XQuery to Java

When a Java function is called from within an XPath/XQuery expression, the datatype of the function's arguments is important in determining which of multiple Java classes having the same name is called.

In Java, the following rules are followed:

- If there is more than one Java method with the same name, but each has a different number of arguments than the other/s, then the Java method that best matches the number of arguments in the function call is selected.
- The XPath/XQuery string, number, and boolean datatypes (*see list below*) are implicitly converted to a corresponding Java datatype. If the supplied XPath/XQuery type can be converted to more than one Java type (for example, `xs:integer`), then that Java type is selected which is declared for the selected method. For example, if the Java method being called is `fx(decimal)` and the supplied XPath/XQuery datatype is `xs:integer`, then `xs:integer` will be converted to Java's `decimal` datatype.

The table below lists the implicit conversions of XPath/XQuery string, number, and boolean types to Java datatypes.

`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.

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).

10.2.2 .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:

- .NET: Constructors
- .NET: Static Methods and Static Fields
- .NET: Instance Methods and Instance Fields
- Datatypes: XPath/XQuery to .NET
- Datatypes: .NET to XPath/XQuery

Form of the extension function
The extension function in the XPath/XQuery expression must have the form `prefix:fname()`.

- The `prefix:` part is associated with a URI that identifies the .NET class being addressed.
- The `fname()` part identifies the constructor, property, or method (static or instance) within the .NET class, and supplies any argument/s, if required.
- The URI must begin with `clitype:` (which identifies the function as being a .NET extension function).
- The `prefix:fname()` form of the extension function can be used with system classes and with classes in a loaded assembly. However, if a class needs to be loaded, additional parameters containing the required information will have to be supplied.

Parameters
To load an assembly, the following parameters are used:

`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.

.NET: Constructors

An extension function can be used to call a .NET constructor. All constructors are called with the pseudo-function `new()`. If there is more than one constructor for a class, then the constructor that most closely matches the number of arguments supplied is selected. If no constructor is deemed to match the supplied argument/s, then a `'No constructor found'` error is returned.

Constructors that return XPath/XQuery datatypes
If the result of a .NET constructor call can be implicitly converted to XPath/XQuery datatypes, then the .NET extension function will return a sequence that is an XPath/XQuery datatype.

Constructors that return .NET objects
If the result of a .NET constructor call cannot be converted to a suitable XPath/XQuery datatype, then the constructor creates a wrapped .NET object with a type that is the name of the class returning that object. For example, if a constructor for the class `System.DateTime` is called (with `System.DateTime.new()`), then an object having a type `System.DateTime` is returned.

The lexical format of the returned object may not match the lexical format of a required XPath datatype. In such cases, the returned value would need to be: (i) converted to the lexical format of the required XPath datatype; and (ii) cast to the required XPath datatype.

There are three things that can be done with a .NET object created by a constructor:

- It can be used within a variable:
  ```
  <xsl:variable name="currentdate" select="date:new(2008, 4, 29)"
  xmlns:date="clitype:System.DateTime" />
  ```
- It can be passed to an extension function (*see Instance Method and Instance Fields*):
  ```
  <xsl:value-of select="date:ToString(date:new(2008, 4, 29))"
  xmlns:date="clitype:System.DateTime" />
  ```
- It can be converted to a string, number, or boolean:
- ```
  <xsl:value-of select="xs:integer(data:get_Month(date:new(2008, 4, 29)))"
  xmlns:date="clitype:System.DateTime" />
  ```

.NET: Static Methods and Static Fields

A static method is called directly by its name and by supplying the arguments for the method. The name used in the call must exactly match a public static method in the class specified. If the method name and the number of arguments that were given in the function call matches more than one method in a class, then the types of the supplied arguments are evaluated for the best match. If a match cannot be found unambiguously, an error is reported.

Note: A field in a .NET class is considered to be a method without any argument. A property is called using the syntax `get_PropertyName()`.

Examples

An XSLT example showing a call to a method with one argument (`System.Math.Sin(arg)`):

```
<xsl:value-of select="math:Sin(30)" xmlns:math="clitype:System.Math"/>
```

An XSLT example showing a call to a field (considered a method with no argument) (`System.Double.MaxValue()`):

```
<xsl:value-of select="double:MaxValue()" xmlns:double="clitype:System.Double"/>
```

An XSLT example showing a call to a property (syntax is `get_PropertyName()`) (`System.String()`):

```
<xsl:value-of select="string:get_Length('my string')"
xmlns:string="clitype:System.String"/>
```

An XQuery example showing a call to a method with one argument (`System.Math.Sin(arg)`):

```
<sin xmlns:math="clitype:System.Math">
   { math:Sin(30) }
</sin>
```

.NET: Instance Methods and Instance Fields

An instance method has a .NET object passed to it as the first argument of the method call. This .NET object typically would be created by using an extension function (for example a constructor call) or a stylesheet parameter/variable. An XSLT example of this kind would be:


```

```
<xsl:stylesheet version="2.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:fn="http://www.w3.org/2005/xpath-functions">
   <xsl:output method="xml" omit-xml-declaration="yes"/>
   <xsl:template match="/">
      <xsl:variable name="releasedate"
         select="date:new(2008, 4, 29)"
         xmlns:date="clitype:System.DateTime"/>
      <doc>
         <date>
            <xsl:value-of select="date:ToString(date:new(2008, 4, 29))"
               xmlns:date="clitype:System.DateTime"/>
         </date>
         <date>
            <xsl:value-of select="date:ToString($releasedate)"
               xmlns:date="clitype:System.DateTime"/>
         </date>
      </doc>
   </xsl:template>
</xsl:stylesheet>
```

In the example above, a `System.DateTime` constructor (`new(2008, 4, 29)`) is used to create a .NET object of type `System.DateTime`. This object is created twice, once as the value of the variable `releasedate`, a second time as the first and only argument of the `System.DateTime.ToString()` method. The instance method `System.DateTime.ToString()` is called twice, both times with the `System.DateTime` constructor (`new(2008, 4, 29)`) as its first and only argument. In one of these instances, the variable `releasedate` is used to get the .NET object.

Instance methods and instance fields

The difference between an instance method and an instance field is theoretical. In an instance method, a .NET object is directly passed as an argument; in an instance field, a parameter or variable is passed instead—though the parameter or variable may itself contain a .NET object. For example, in the example above, the variable `releasedate` contains a .NET object, and it is this variable that is passed as the argument of `ToString()` in the second `date` element constructor. Therefore, the `ToString()` instance in the first `date` element is an instance method while the second is considered to be an instance field. The result produced in both instances, however, is the same.

Datatypes: XPath/XQuery to .NET

When a .NET extension function is used within an XPath/XQuery expression, the datatypes of the function's arguments are important for determining which one of multiple .NET methods having the same name is called.

In .NET, the following rules are followed:

- If there is more than one method with the same name in a class, then the methods

available for selection are reduced to those that have the same number of arguments as the function call.
- The XPath/XQuery string, number, and boolean datatypes (*see list below*) are implicitly converted to a corresponding .NET datatype. If the supplied XPath/XQuery type can be converted to more than one .NET type (for example, `xs:integer`), then that .NET type is selected which is declared for the selected method. For example, if the .NET method being called is `fx(double)` and the supplied XPath/XQuery datatype is `xs:integer`, then `xs:integer` will be converted to .NET's `double` datatype.

The table below lists the implicit conversions of XPath/XQuery string, number, and boolean types to .NET datatypes.

`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.

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).

10.2.3 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.

10.2.4 MSXSL Scripts for XSLT

The `<msxsl:script>` element contains user-defined functions and variables that can be called from within XPath expressions in the XSLT stylesheet. The `<msxsl:script>` is a top-level element, that is, it must be a child element of `<xsl:stylesheet>` or `<xsl:transform>`.

The `<msxsl:script>` element must be in the namespace `urn:schemas-microsoft-com:xslt` (*see example below*).

Scripting language and namespace
The scripting language used within the block is specified in the `<msxsl:script>` element's `language` attribute and the namespace to be used for function calls from XPath expressions is identified with the `implements-prefix` attribute (*see below*).

```
<msxsl:script language="scripting-language" implements-prefix="user-namespace-
prefix">

   function-1 or variable-1
   ...
   function-n or variable-n

</msxsl:script>
```

The `<msxsl:script>` element interacts with the Windows Scripting Runtime, so only languages that are installed on your machine may be used within the `<msxsl:script>` element. **The .NET Framework 2.0 platform or higher must be installed for MSXSL scripts to be used.** Consequently, the .NET scripting languages can be used within the `<msxsl:script>` element.

The `language` attribute accepts the same values as the `language` attribute on the HTML `<script>` element. If the `language` attribute is not specified, then Microsoft JScript is assumed as the default.

The implements-prefix attribute takes a value that is a prefix of a declared in-scope namespace. This namespace typically will be a user namespace that has been reserved for a function library. All functions and variables defined within the `<msxsl:script>` element will be in the namespace identified by the prefix specified in the `implements-prefix` attribute. When a function is called from within an XPath expression, the fully qualified function name must be in the same namespace as the function definition.

Example
Here is an example of a complete XSLT stylesheet that uses a function defined within a `<msxsl:script>` element.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
```

```
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:fn="http://www.w3.org/2005/xpath-functions"
    xmlns:msxsl="urn:schemas-microsoft-com:xslt"
    xmlns:user="http://mycompany.com/mynamespace">

  <msxsl:script language="VBScript" implements-prefix="user">
    <![CDATA[
    ' Input: A currency value: the wholesale price
    ' Returns: The retail price: the input value plus 20% margin,
    ' rounded to the nearest cent
    dim a as integer  = 13
    Function AddMargin(WholesalePrice) as integer
      AddMargin = WholesalePrice * 1.2 + a
    End Function
  ]]>
  </msxsl:script>

  <xsl:template match="/">
    <html>
      <body>
        <p>
          <b>Total Retail Price =
            $<xsl:value-of select="user:AddMargin(50)"/>
          </b>
          <br/>
          <b>Total Wholesale Price =
            $<xsl:value-of select="50"/>
          </b>
        </p>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Datatypes

The values of parameters passed into and out of the script block are limited to XPath datatypes. This restriction does not apply to data passed among functions and variables within the script block.

Assemblies

An assembly can be imported into the script by using the **msxsl:assembly** element. The assembly is identified via a name or a URI. The assembly is imported when the stylesheet is compiled. Here is a simple representation of how the `msxsl:assembly` element is to be used.

```
<msxsl:script>
    <msxsl:assembly name="myAssembly.assemblyName" />
    <msxsl:assembly href="pathToAssembly" />
```

```
    ...
</msxsl:script>
```

The assembly name can be a full name, such as:

```
"system.Math, Version=3.1.4500.1 Culture=neutral
PublicKeyToken=a46b3f648229c514"
```

or a short name, such as `"myAssembly.Draw"`.

Namespaces

Namespaces can be declared with the **`msxsl:using`** element. This enables assembly classes to be written in the script without their namespaces, thus saving you some tedious typing. Here is how the `msxsl:using` element is used so as to declare namespaces.

```
<msxsl:script>
    <msxsl:using namespace="myAssemblyNS.NamespaceName" />

    ...
</msxsl:script>
```

The value of the `namespace` attribute is the name of the namespace.

Chapter 11

Altova LicenseServer

11 Altova LicenseServer

Altova LicenseServer (hereafter also called **LicenseServer** for short) 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.

Current version: 1.11

Licensing process with Altova LicenseServer
To assign an Altova server product a license via Altova LicenseServer, you need to do the following:

1. Start LicenseServer
2. Open the LicenseServer Configuration page, which is the Web UI of LicenseServer, on Windows, Linux, or Mac OS X.
3. Upload the license/s you have received from Altova to LicenseServer. Do this in the License Pool tab of the Configuration page.
4. Register Altova server products (FlowForce Server, MapForce Server, StyleVision Server, RaptorXML(+XBRL) Server) with LicenseServer.
5. Assign licenses to Altova server In the Server Management tab of the Configuration page.

Licenses can thereafter be conveniently monitored and managed centrally with LicenseServer. See the Configuration Page Reference for available functionality.

Note: The LicenseServer Configuration page does not support SSL.

▼ **LicenseServer versions and their compatibility with Altova server products**
New versions of Altova server products can only be licensed with the version of LicenseServer that is the latest at the time of the server product's release. However, older versions of Altova server products will work with newer versions of LicenseServer.

So, if you are installing a new version of an Altova server product and if your current LicenseServer version is not the latest, de-install this older version and install the latest version available on the Altova website. All registration and licensing information held in your older version of LicenseServer will be saved at the time of de-installation to a database on your server machine, and will be imported automatically into the newer version. When you install a newer version of LicenseServer, the older version will be de-installed before the newer version is installed.

The version number of the currently installed LicenseServer is given at the bottom of the LicenseServer configuration page (all tabs).

Current version: 1.11

About this documentation

This documentation is organized into the following parts:

- Introductory information about: network requirements; installation on Windows, Linux, and Mac OS X; 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.

Last updated: 09-15-2014

11.1 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 machine, **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.

11.2 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, Altova StyleVision Server, and Altova RaptorXML(+XBRL).)

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.

For information about how to proceed with assigning licenses, see the section How to Assign Licenses.

LicenseServer versions and their compatibility with Altova server products

New versions of Altova server products can only be licensed with the version of LicenseServer that is the latest at the time of the server product's release. However, older versions of Altova server products will work with newer versions of LicenseServer.

So, if you are installing a new version of an Altova server product and if your current LicenseServer version is not the latest, de-install this older version and install the latest version available on the Altova website. All registration and licensing information held in your older version of LicenseServer will be saved at the time of de-installation to a database on your server machine, and will be imported automatically into the newer version. When you install a newer version of LicenseServer, the older version will be de-installed before the newer version is installed.

The version number of the currently installed LicenseServer is given at the bottom of the LicenseServer configuration page (all tabs).

Current version: 1.11

The version number of the LicenseServer that is appropriate for any particular version of a server product is displayed during the installation of that version of the server product. You can choose to install this version of LicenseServer along with the server product, or you can install the newer version of LicenseServer separately. In both cases, the installer will automatically de-install the previous version and install the new version.

11.3 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-1.11-debian.deb
[Ubuntu]:   sudo dpkg --install licenseserver-1.11-ubuntu.deb
[CentOS]:   sudo rpm -ivh licenseserver-1.11-1.x86_64.rpm
[RedHat]:   sudo rpm -ivh licenseserver-1.11-1.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.

- **LicenseServer versions and their compatibility with Altova server products**
  New versions of Altova server products can only be licensed with the version of LicenseServer that is the latest at the time of the server product's release. However, older versions of Altova server products will work with newer versions of LicenseServer.

  So, if you are installing a new version of an Altova server product and if your current LicenseServer version is not the latest, de-install this older version and install the latest version available on the Altova website. All registration and licensing information held in your older version of LicenseServer will be saved at the time of de-installation to a database on your server machine, and will be imported automatically into the newer version. When you install a newer version of LicenseServer, the older version will be de-installed before the newer version is installed.

  The version number of the currently installed LicenseServer is given at the bottom of the LicenseServer configuration page (all tabs).

  Current version: 1.11

11.4 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.LicenseServer.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:

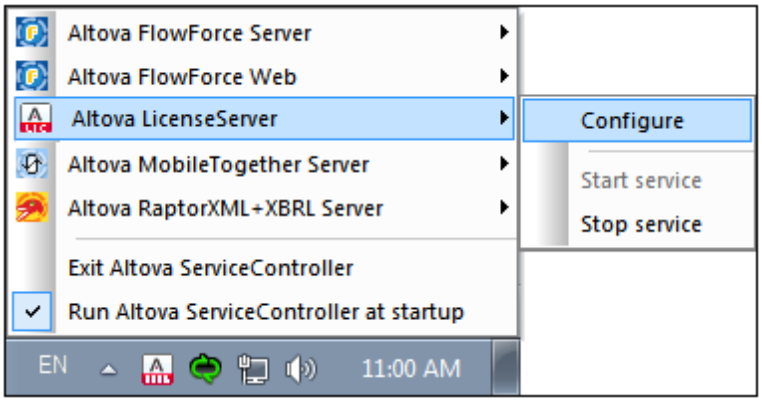
```
/usr/local/Altova/LicenseServer
```


11.5 Altova ServiceController

Altova ServiceController (ServiceController for short) is an application for conveniently starting, stopping and configuring Altova services **on Windows systems**.

ServiceController is installed with Altova LicenseServer and with *Altova server products that are installed as services* (FlowForce Server, RaptorXML(+XBRL) Server, and Mobile Together Server). It can be started by clicking **Start | Altova LicenseServer | Altova ServiceController**. (This command is also available in the **Start** menu folders of *Altova server products that are installed as services* (FlowForce Server, RaptorXML(+XBRL) Server, and Mobile Together Server).) After ServiceController has been started, it can be accessed via the system tray (*screenshot below*).

To specify that ServiceController starts automatically on logging in to the system, click the **ServiceController** icon in the system tray to display 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 appears (*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, Altova LicenseServer service has a sub-menu in which you can choose to access LicenseServer's Configuration page via the **Configure** command.

11.6 How to Assign Licenses

To assign an Altova server product a license using Altova LicenseServer, do the following:

1. Start LicenseServer
2. Open the LicenseServer Configuration page, which is the administrator's interface with LicenseServer, on Windows, Linux, or Mac OS X.
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, RaptorXML(+XBRL) Server) with LicenseServer. Depending on the product's type, the method of registering it with LicenseServer will be different: either via the product's Web UI or its command line. See the documentation of your Altova server product for additional information.
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, *except MobileTogether Server****, 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.

*** MobileTogether Server licenses are assigned on the basis of the number of users, that is, the number of client devices that connect to MobileTogether Server.

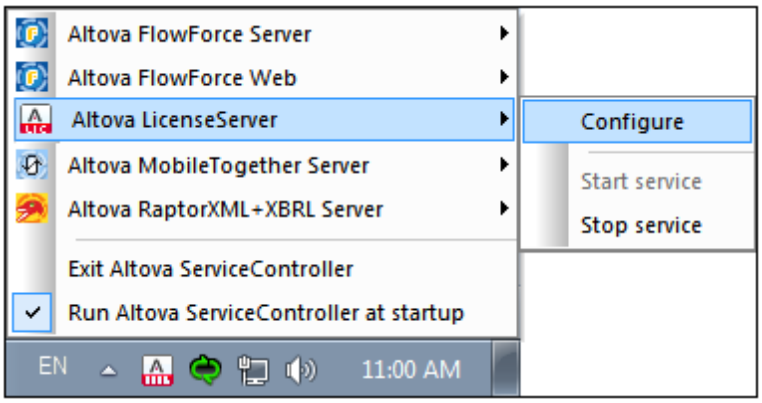
11.6.1 Start LicenseServer

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.

First, click **Start | All Programs | Altova LicenseServer | Altova ServiceController** to start Altova ServiceController and display its icon in the system tray (*see screenshot below*). If you select the *Run Altova ServiceController at Startup* option, 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, hover 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.)

Mac OS X systems
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.LicenseServer.plist
```

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

```
sudo launchctl unload /Library/LaunchDaemons/com.altova.LicenseServer.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.

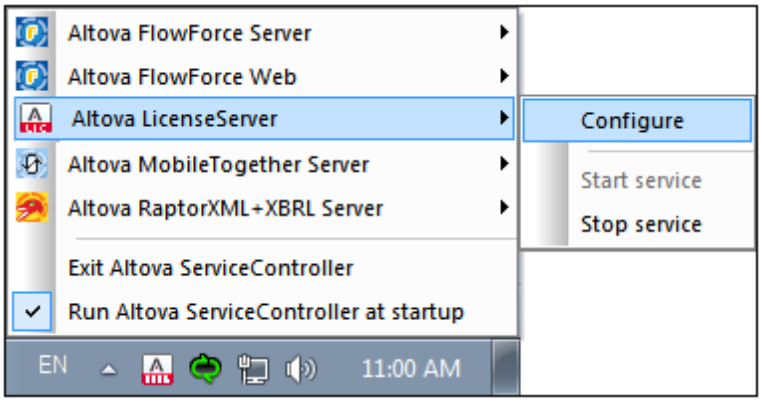
Any such loss of connection with the `altova.com` master servers will be logged in the Messages tab of the Configuration page of the Altova LicenseServer. In addition, the administrator can configure the Altova LicenseServer to automatically send an alert email when the connection to `altova.com` is lost. Alert Mail settings are available in the Settings tab of the Configuration page.

11.6.2 Open LicenseServer's 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/
```

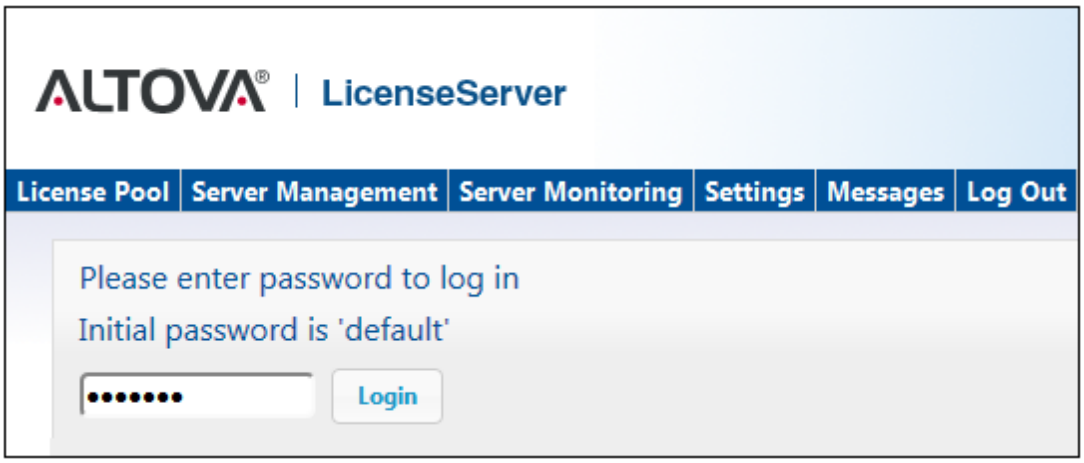
The URL is present in the HTML code of the Configuration page itself, which is named `WebUI.html` and is located at:

```
C:/ProgramData/Altova/LicenseServer/WebUI.html
```

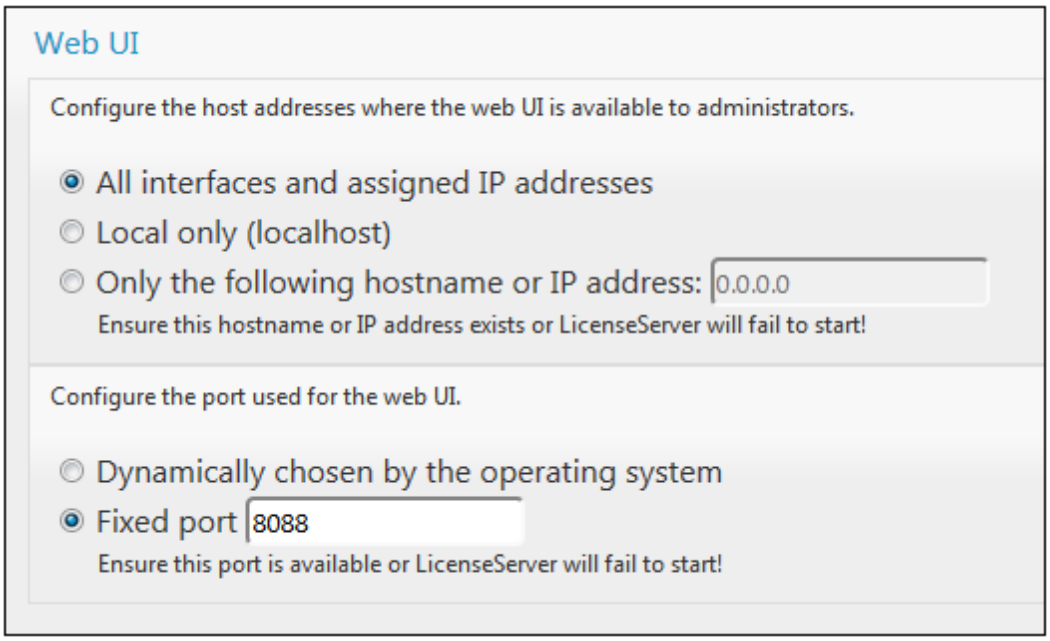
If you have set the URL of the Configuration page to be generated dynamically (in the Settings tab of the Configuration page), then a new URL is generated each time LicenseServer is started. You will need to check the current version of **WebUI.html** to find out the current URL of the Configuration 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 (Web UI)—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.

11.6.3 Open LicenseServer's 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/
```

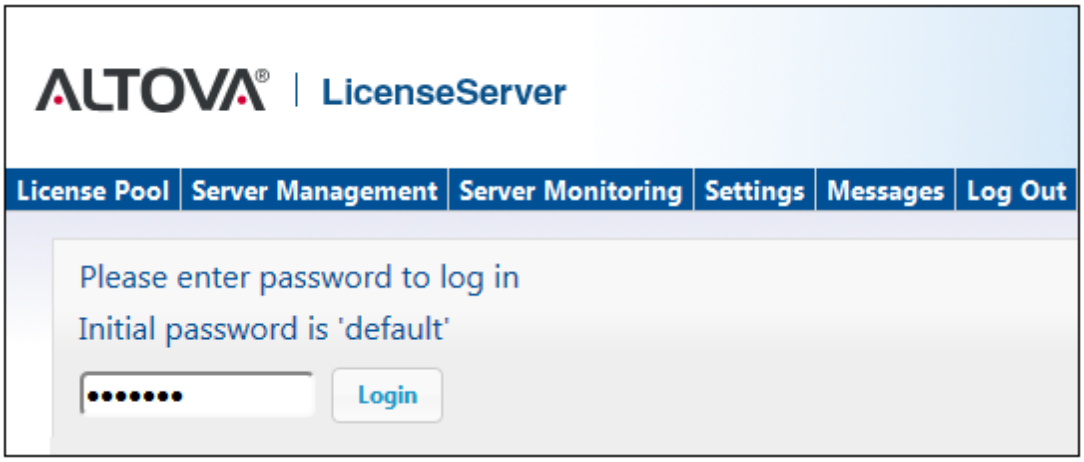
The URL is present in the HTML code of the Configuration page itself, which is named **webUI.html** and is located at:

```
/var/opt/Altova/LicenseServer/webUI.html
```

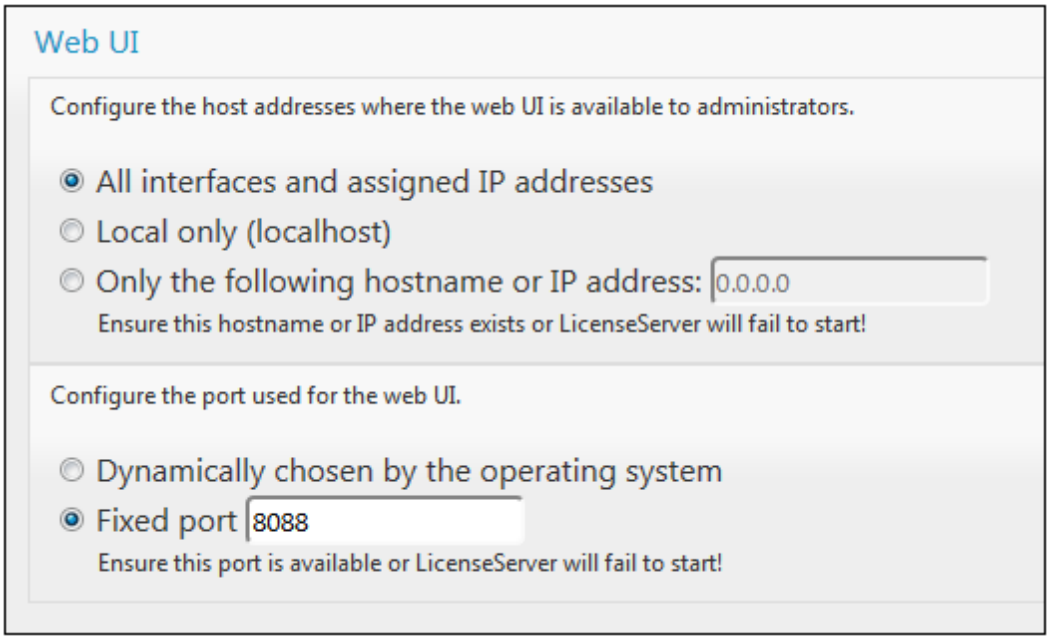
If you have set the URL of the Configuration page to be generated dynamically (in the Settings tab of the Configuration page), then a new URL is generated each time LicenseServer is started. You will need to check the current version of **webUI.html** to find out the current URL of the Configuration 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 (Web UI)—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.

11.6.4 Open LicenseServer's 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
```

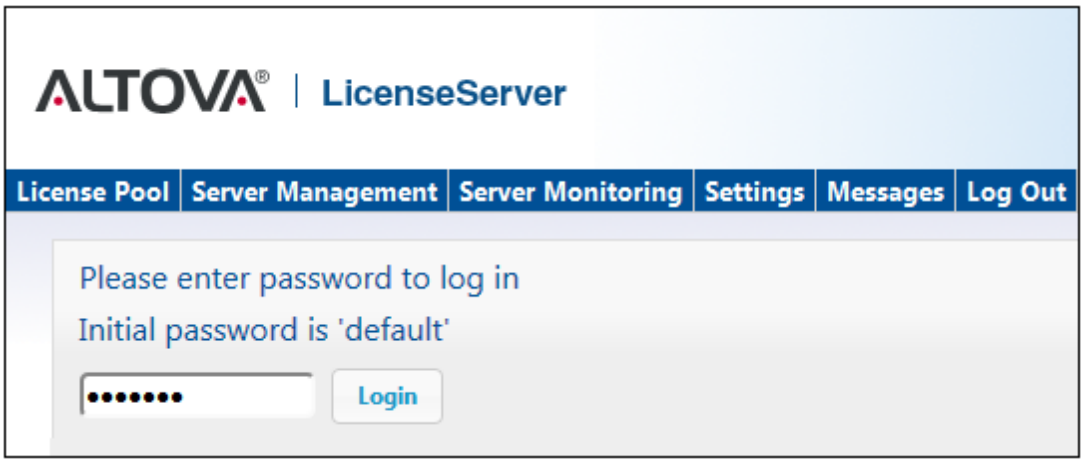
If you have set the URL of the Configuration page to be generated dynamically (in the Settings tab of the Configuration page), then a new URL is generated each time LicenseServer is started. You will need to check the current version of **webUI.html** to find out the current URL of the Configuration 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`.

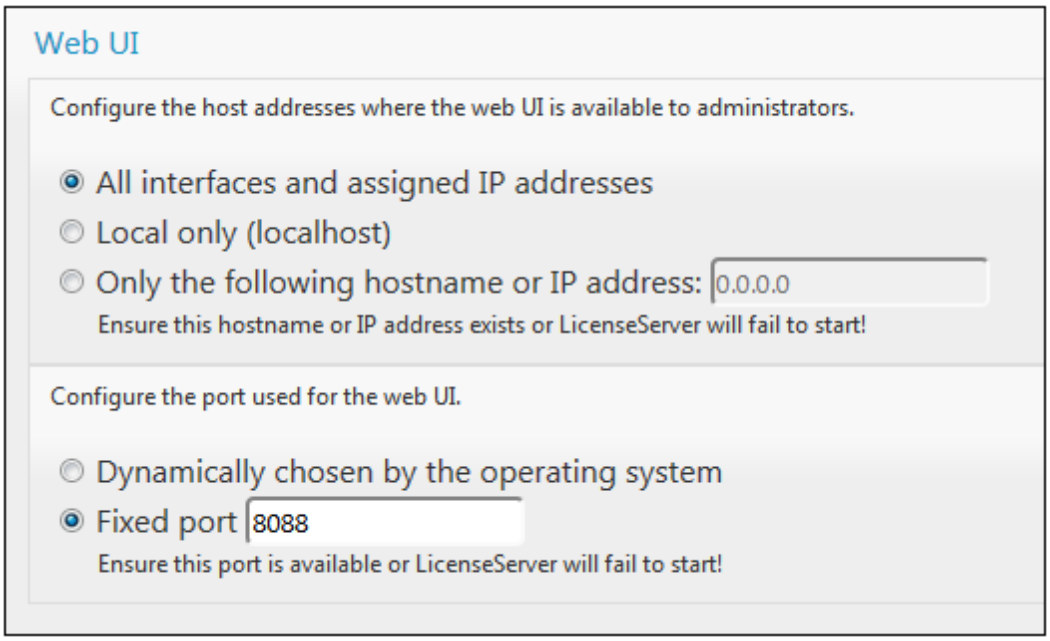
Note: The Configuration page can also be accessed directly via the **Finder | Applications | Altova License Server** icon.

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 (Web UI)—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

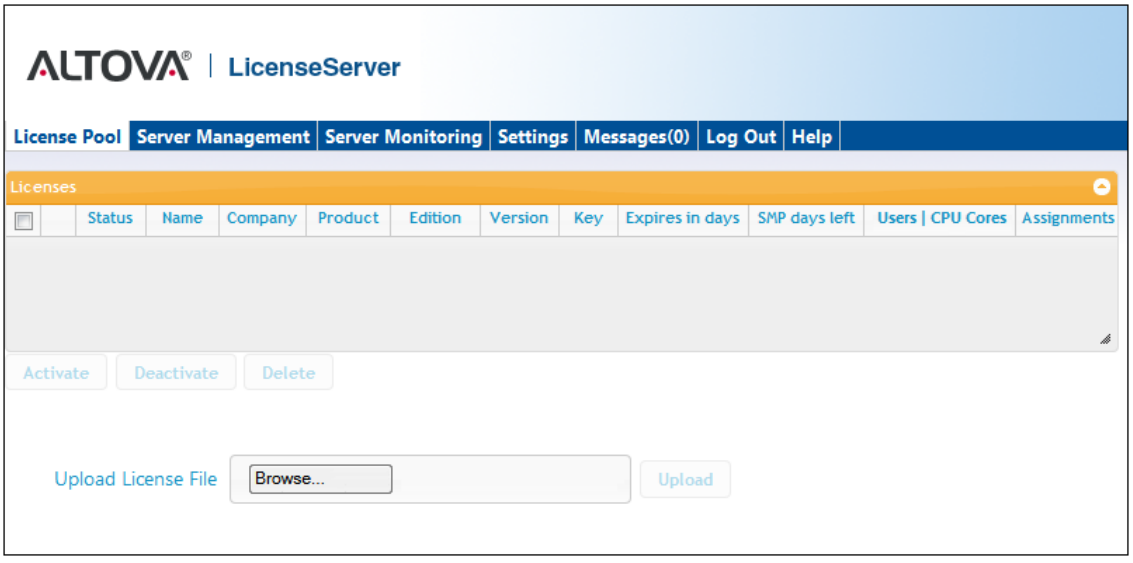
accessed easily. If the port is assigned dynamically, the port part of the URL will have to be looked up in the file `WebUI.html` each time LicenseServer is started.

11.6.5 Upload Licenses to LicenseServer

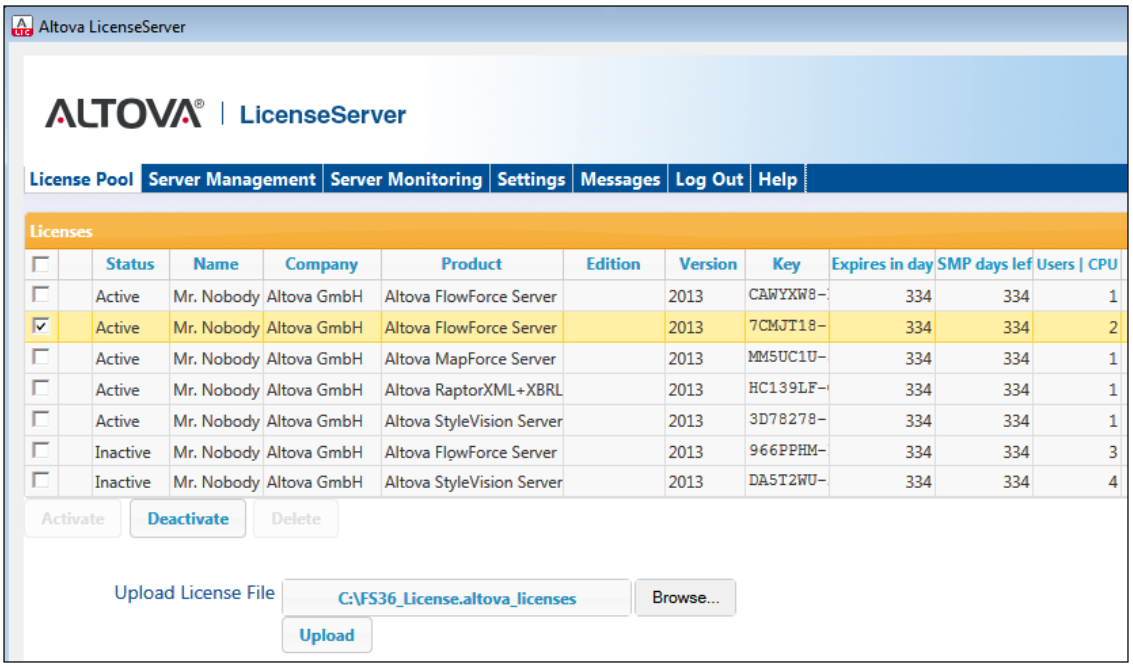
This section:

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 LicenseServer 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.

11.6.6 Register Product/s

Before you can assign a license to an Altova server product, you must register the product installation with LicenseServer. The registration is done from the Altova server product, and the process is different for those server products that have Web UIs and those that are run from the command line only. You will need the server name or IP Address of the machine on which LicenseServer is installed to carry out the registration.

This section describes how to register different Altova server products:

- Register FlowForce Server
- Register MapForce Server
- Register StyleVision Server
- Register RaptorXML(+XBRL) Server
- Register MobileTogether Server

Register FlowForce Server

This section:

- Methods of registering FlowForce Server with LicenseServer
- Accessing the FlowForce Server Setup page (Windows)
- Accessing the FlowForce Server Setup page (Linux)
- Registering FlowForce Server via the Setup page
- Registering FlowForce Server via the FlowForce CLI (Windows)
- Registering FlowForce Server via the FlowForce CLI (Linux)
- Next steps

Methods of registering FlowForce Server
FlowForce Server can be registered with LicenseServer using any of the following methods:

- Via the FlowForce Server Setup page
- 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 2015* | *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:

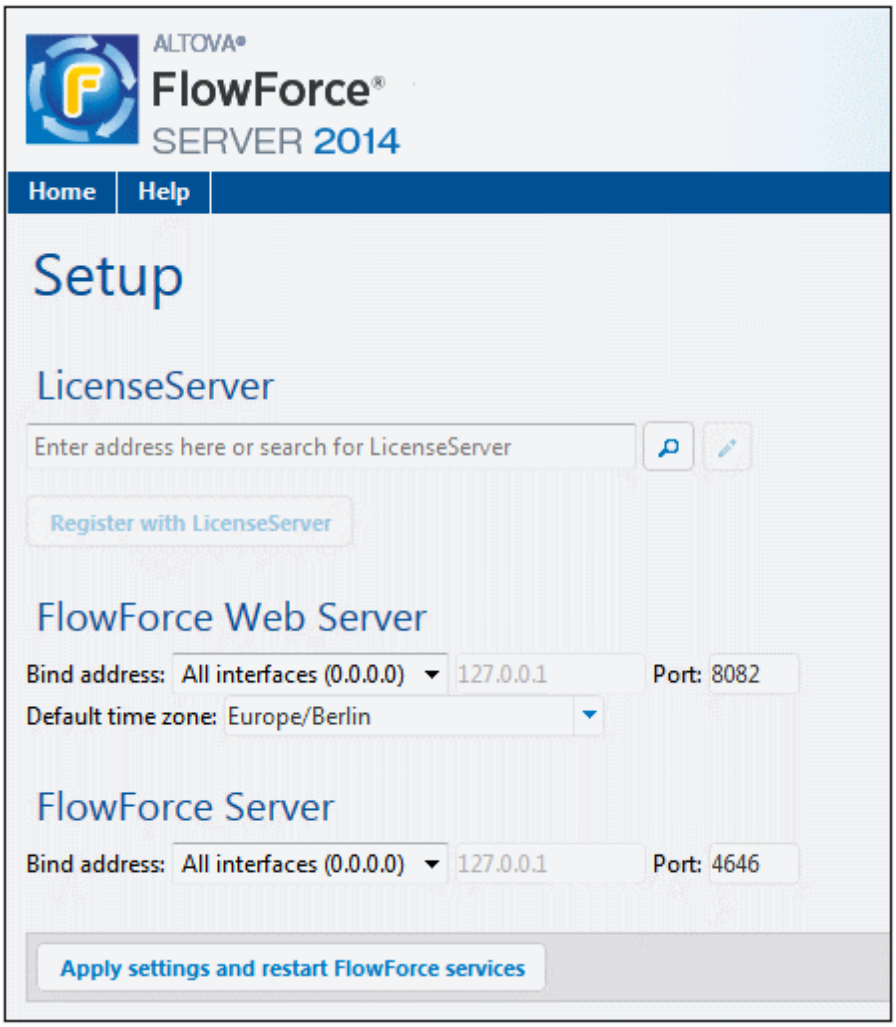
```
sudo /etc/init.d/flowforcewebserver start
```

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
```

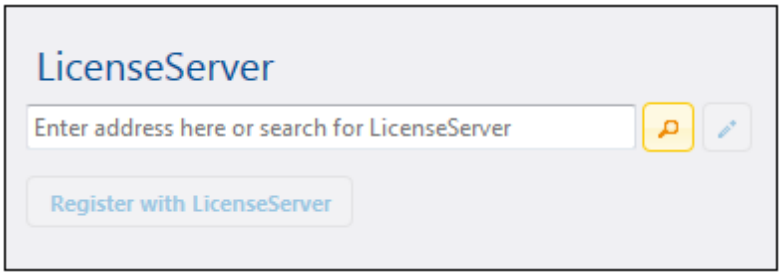
Enter the URL in the address field of a browser and hit **Enter** to access the FlowForce Server Setup page (*screenshot below*).

Registering FlowForce Server via the Setup page
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.

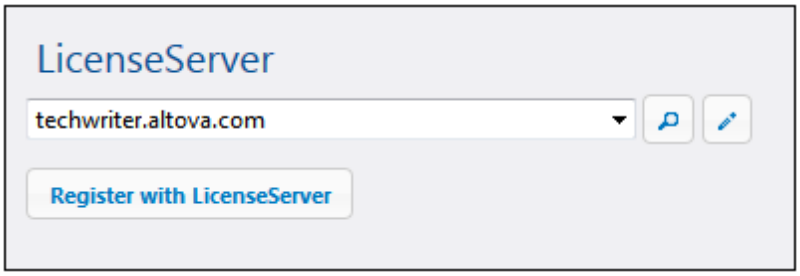


The LicenseServer can be specified in one of two ways.

- 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*).



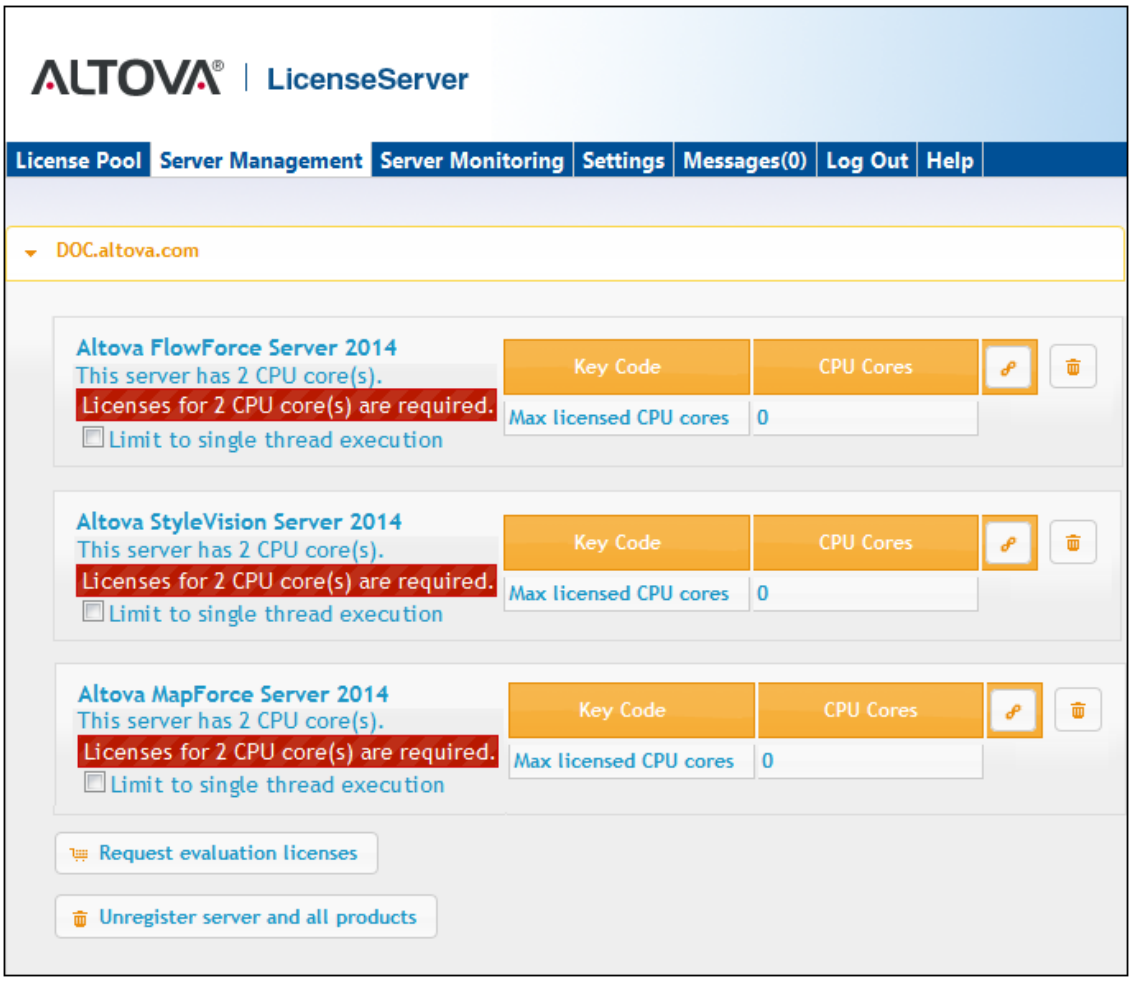
The search returns a list of available Altova LicenseServers on the network. One LicenseServer will be selected (*screenshot below*) and the others will be available in the dropdown list of the combo box. Select the LicenseServer on which your FlowForce license is stored.



- 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/FlowForceServer2015/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/MapForceServer2015/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 the license/s), 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.

Register MapForce Server

This section:

- Registering MapForce Server from FlowForce Server (Windows)
- Registering a standalone MapForce Server (Windows)
- Registering MapForce Server (Linux)
- Next steps

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/MapForceServer2015/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/MapForceServer2015/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 the license/s), 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.

Register StyleVision Server

This section:

- Registering StyleVision Server from FlowForce Server (Windows)
- Registering a standalone StyleVision Server (Windows)
- Registering StyleVision Server (Linux)
- Next steps

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/StyleVisionServer2015/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/StyleVisionServer2015/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 the license/s), 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.

Register RaptorXML(+XBRL) Server

This section:

RaptorXML(+XBRL) Server must be installed on the server machine or network to which LicenseServer is connected and then be started as a service. It must then be registered with LicenseServer. Only after registration 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 Server:	**`RaptorXML licenseserver Server-Or-IP-Address`**
RaptorXML+XBRL Server:	**`RaptorXMLXBRL licenseserver Server-Or-IP-Address`**

For example, if LicenseServer is running on `http://localhost:8088`, then register RaptorXML(+XBRL) Server with:

RaptorXML Server:	**`RaptorXML licenseserver localhost`**
RaptorXML+XBRL Server:	**`RaptorXMLXBRL 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/RaptorXMLServer2015/bin/raptorxmlserver licenseserver
localhost
```



```
sudo /opt/Altova/RaptorXMLXBRLServer2015/bin/raptorxmlxbrlserver 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/RaptorXMLServer2015/bin
/opt/Altova/RaptorXMLXBRLServer2015/bin
```

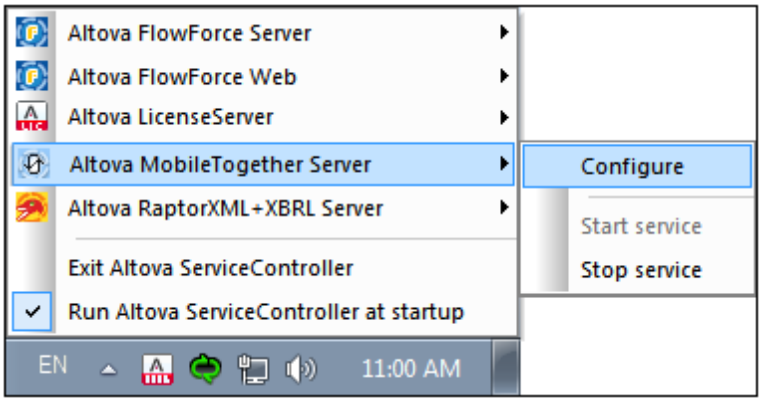
After successfully registering RaptorXML(+XBRL) Server, you can go to LicenseServer and assign a license to RaptorXML(+XBRL) Server. How to do this is described in the section Assign Licenses to Registered Products.

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 the license/s), 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.

Register MobileTogether Server

To start MobileTogether Server, click the **ServiceController** icon in the system tray, hover over **Altova MobileTogether Server** in the menu that pops up (*see screenshot below*), and then select **Start Service** from the MobileTogether Server submenu. If MobileTogether Server is already running, the *Start Service* option will be disabled.



Register MobileTogether Server via:

- The Settings tab of the MobileTogether Server Web UI: (i) Start MobileTogether Server via ServiceController (*see previous point*); (ii) Enter your password to access the Setup page; (iii) Select the LicenseServer name or address, and click **Register with LicenseServer**.
- its CLI, using the `licenseserver` command:

MobileTogetherServer licenseserver [options] ***ServerName-Or-IP-Address***
For example, if `localhost` is the name of the server on which LicenseServer is installed:
MobileTogetherServer licenseserver localhost

After successful registration, go to the Server Management tab of LicenseServer's configuration page to assign a license to MobileTogether Server.

11.6.7 Assign Licenses to Registered Products

This section:

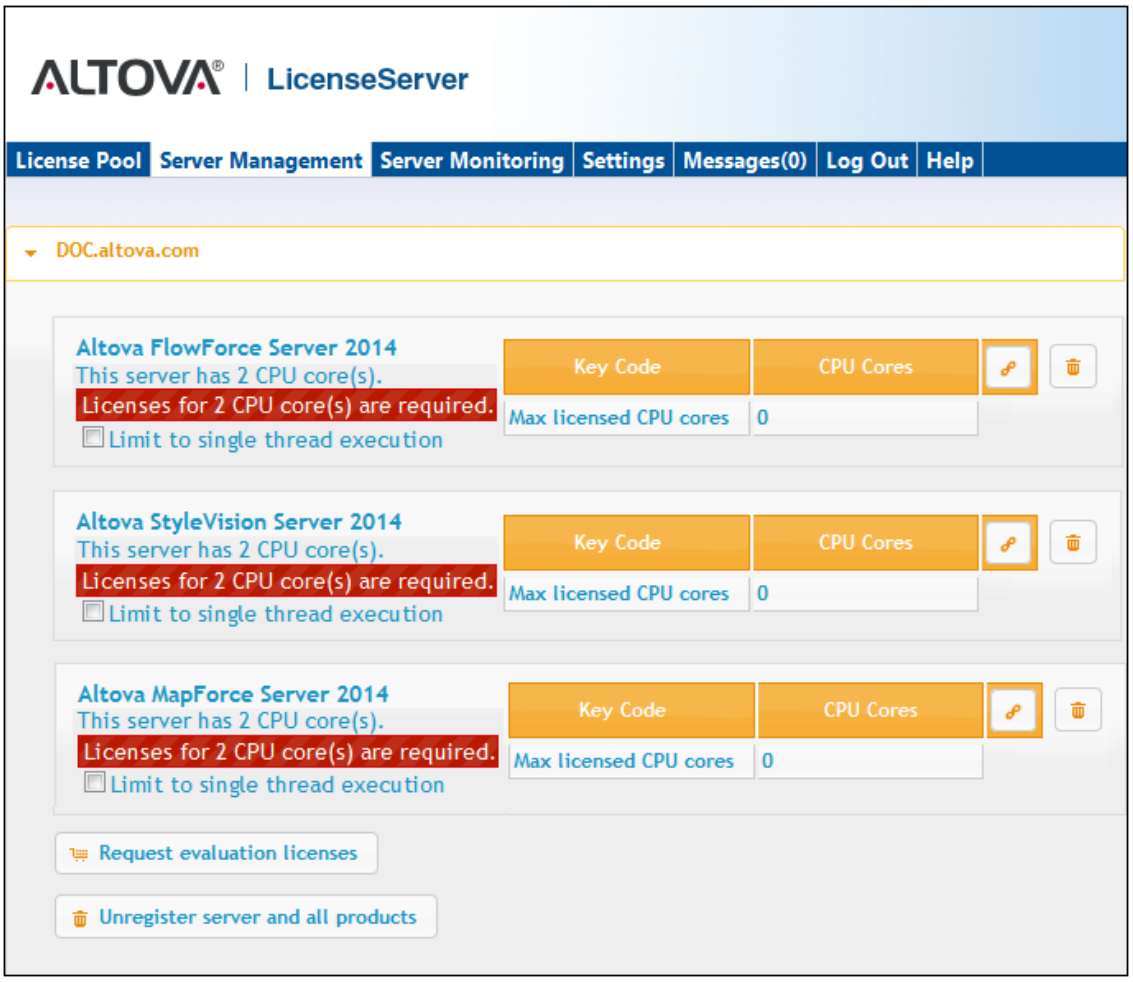
Before assigning a license

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.)
- If **multiple versions** of a single product (for example, StyleVision Server 2013 and StyleVision Server 2014) have been installed on one machine and if each of these installations has been registered with a single LicenseServer, then the multiple registrations are consolidated in a single registration in the Server Management tab and displayed as a single registration. When a license is assigned to this single registration,

all the installations indicated by that registration will be licensed. However, multiple instances of only one installation can be run simultaneously on the client machine. For example, multiple instances of StyleVision Server 2013 or multiple instances of StyleVision Server 2014 can be run simultaneously, but not one instance of StyleVision Server 2013 and one instance of StyleVision Server 2014. Note that newly installed versions must be registered for them to run.

- New versions of Altova server products can only be licensed with the latest version of LicenseServer at the time of the product's release. Older Altova server products will work with newer versions of LicenseServer. So, if you are installing a new version of an Altova server product and if your current LicenseServer version is not the latest, de-install the older version of LicenseServer and install the latest version. All registration and licensing information held in your older version of LicenseServer will be saved, at the time of de-installation, to a database on the server, and will be imported automatically into the newer version. (The version number of the LicenseServer that is appropriate for any particular version of a server product is displayed during the installation of that server product. You can choose to install this version along with the server product. The version of the currently installed LicenseServer is given at the bottom of the LicenseServer configuration page.)

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, *except MobileTogether Server****, 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.

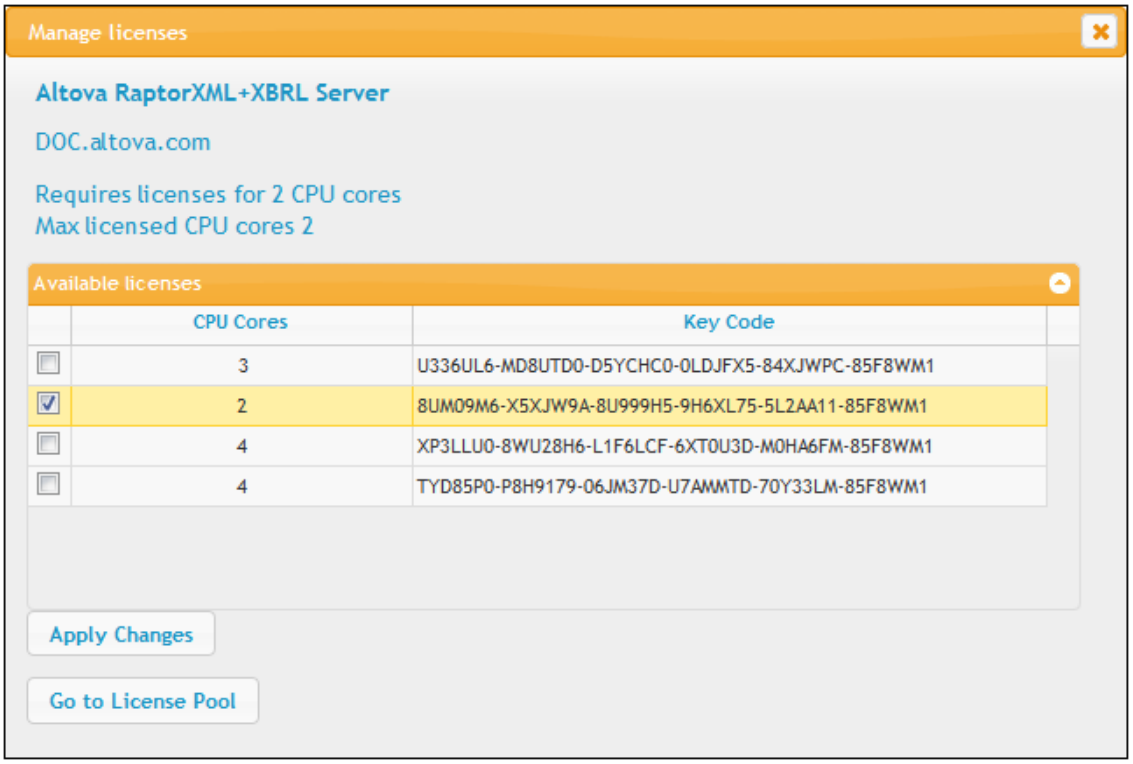
For example, if a server has eight cores (an octa-core processor), you must purchase at least an 8-core license. You can also combine licenses to achieve the core count. So, two 4-core licenses can also be used for an octa-core server instead of an 8-core license.

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.

*** MobileTogether Server licenses are assigned on the basis of the number of users, that is, the number of client devices that connect to MobileTogether Server.

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:

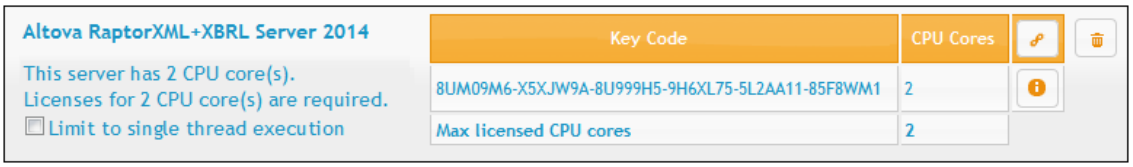
- The product to be licensed is listed at the top left of the dialog. In the screenshot above the product is Altova FlowForce Server 2013.
- The dialog displays all the currently active licenses for that product in the license pool. In our screenshot, four currently active FlowForce Server licenses are 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.

11.7 Configuration Page Reference

The LicenseServer Configuration page is the administrator's interface with LicenseServer (Web UI). It allows the management of LicenseServer and the licensing of Altova products that have been registered with LicenseServer (FlowForce Server, MapForce Server, StyleVision Server, RaptorXML(+XBRL) Server). The LicenseServer Configuration page is viewed in 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 reference for the Configuration page and is organized by the tabs of the Configuration page:

- License Pool
- Server Management
- Server Monitoring
- Settings
- Messages, Log Out

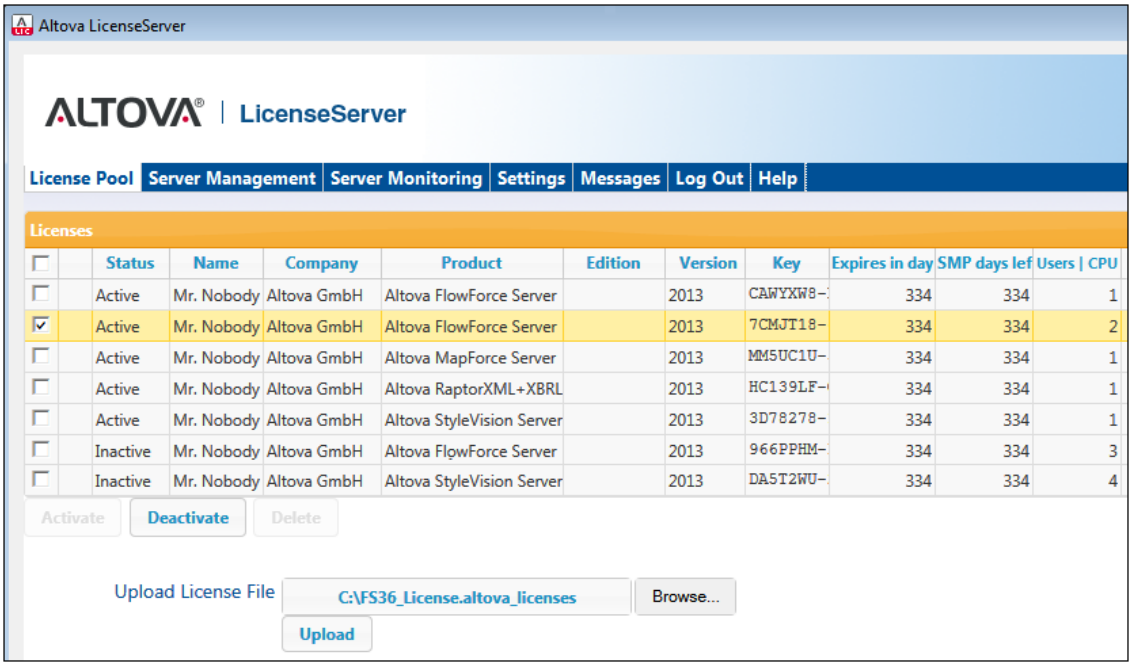
For a step-by-step guide of how to assign licenses with LicenseServer, see the section How to Assign Licenses.

11.7.1 License Pool

This section:

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 server 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.

Icons in the License Pool tab

Edit Assigned Licenses. Appears with each license (in the Assignments column). Pops up the Manage Licenses dialog, in which new licenses can be assigned to the product and already assigned licenses can be edited.

Show License Information. Appears with each license (in the Assignments column). Provides information about the currently active clients.

License information
The following license information is displayed:

- *Status:* Can be one of the following 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.
- *Users* | *CPU Cores:* The number of users or CPU cores that the license allows. In the case of Altova's MobileTogether Server product, licenses are assigned on the basis of the number of MobileTogether client devices that connect to MobileTogether Server. In the case of all other Altova server products, licenses are assigned on the basis of CPU cores (*see note below*).
- *Assignments:* Access to editing dialogs and information of individual licenses.

Note on cores and licenses
The licensing of Altova server products, *except MobileTogether Server****, 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.

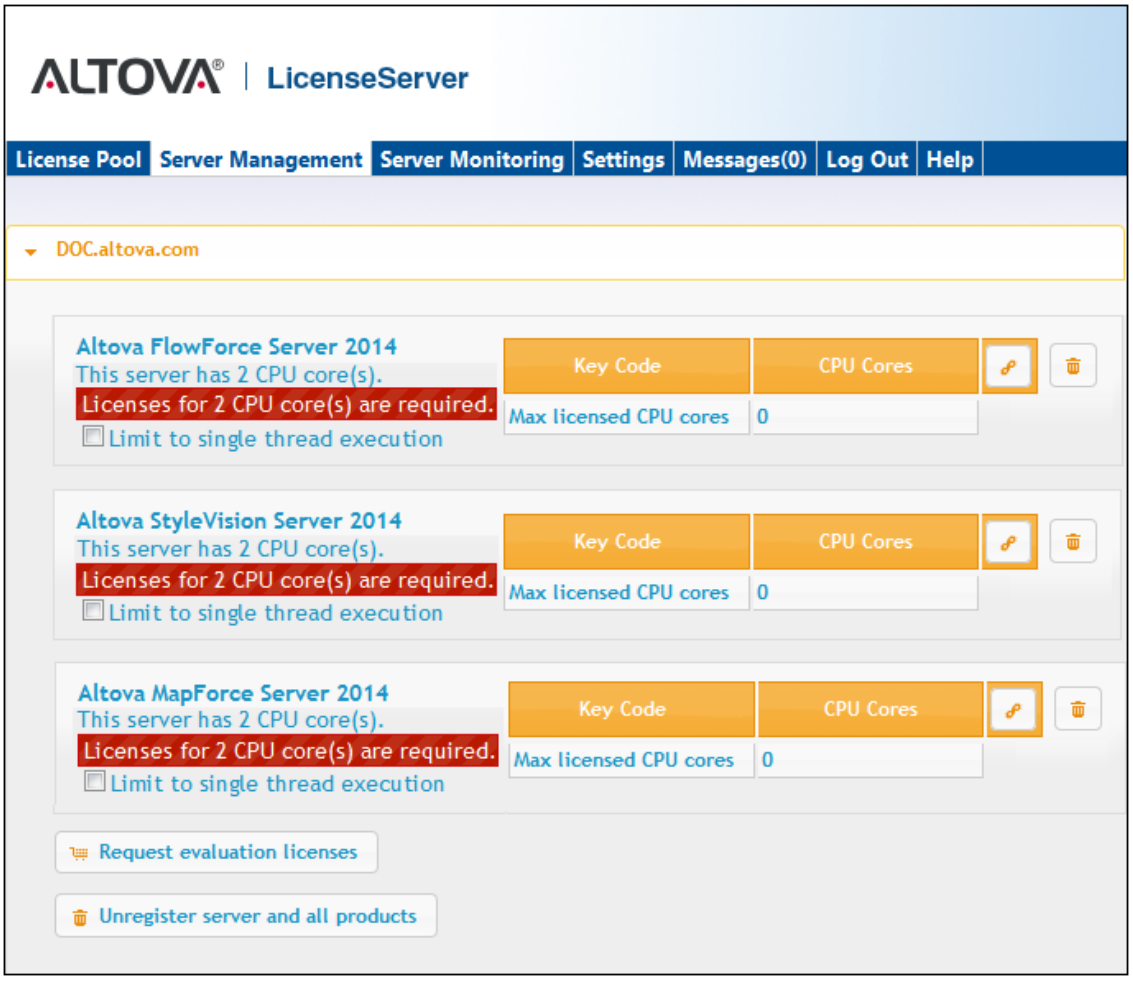
*** MobileTogether Server licenses are assigned on the basis of the number of users, that is, the number of client devices that connect to MobileTogether Server.

11.7.2 Server Management

This section:

- Icons in the Server Management tab
- Assigning licenses
- One client machine under different names
- Requesting an evaluation license
- Unregistering products

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 (except MobileTogether Server) 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.)

Single thread execution
If a product license for only one core is available in the license pool, a machine with multiple cores can be assigned this one-core license. In such a case, the machine will run that product on a single core. Processing will therefore be slower as multi-threading (which is possible on multiple cores) will not be available. The product will be executed in single thread mode on that machine.

To assign a single-core license to a multiple-core machine, select the *Limit to single thread execution* check box for that product.

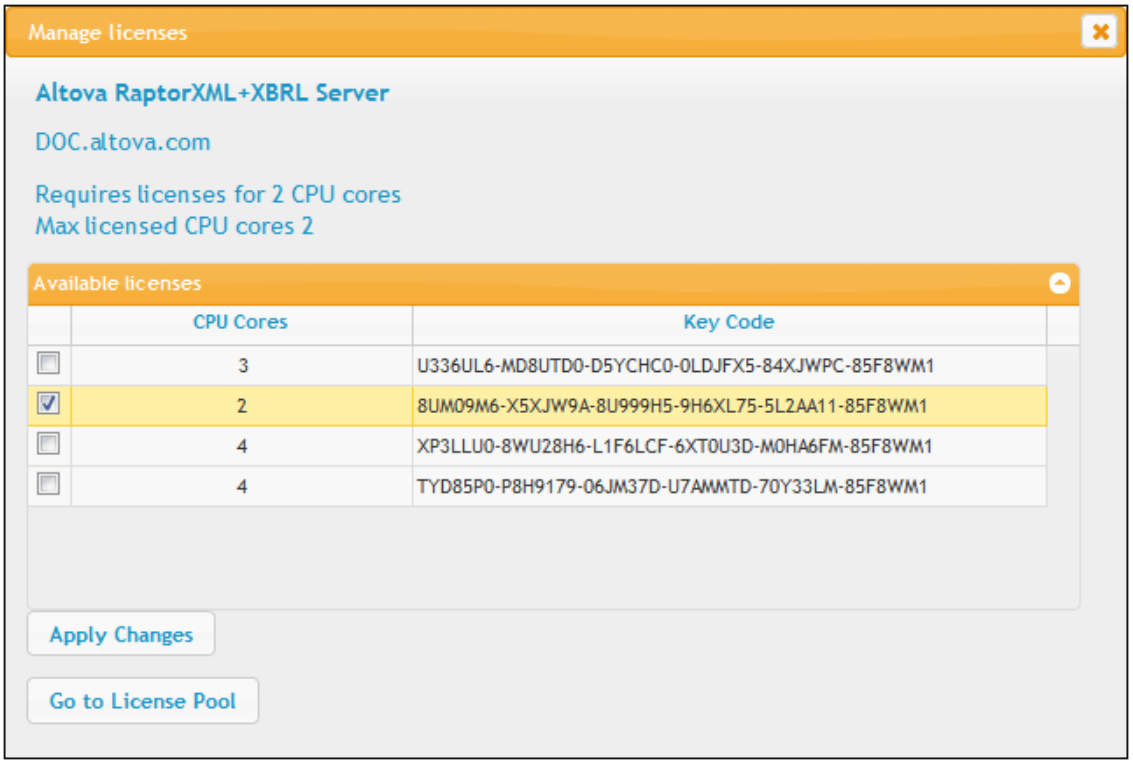
Icons in the Server Management tab

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.

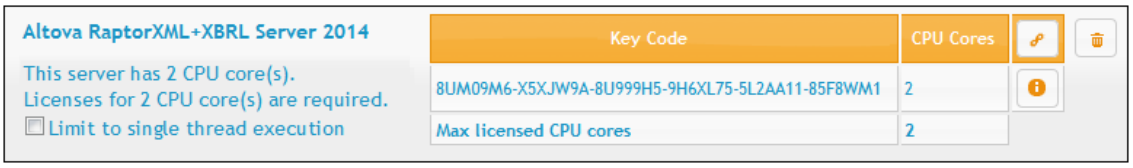
Show Licenses. Appears with each license. Switches to the License Pool tab and highlights the selected license, so that license details can be read.

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*).



One client machine under different names
If a client machine is registered more than once with LicenseServer, it might appear in the Server Management tab under multiple names, that is, with multiple entries. This could happen, for example, if a machine is re-registered with the host name given in a different form.

To ensure that additional licenses are not redundantly assigned to the same machine under its different names, you should unregister redundant client machine entries by clicking the **Unregister server and all products** button of these machines. *(Note: While the client machines are considered for the purposes of this documentation to be clients of LicenseServer, they are in effect servers of their own products.)* Also, if the same license is assigned multiple times to the same machine under its different names, licensing conflicts could arise. So, to avoid these two situations (redundant licensing and multiple assignments of a single license), it is recommended that redundant entries of a single client machine be unregistered.

Given below are forms a machine name might take in the Server Management tab:

- *Host name with domain name (the fully qualified domain name, FQDN), such as: "win80-x64_1.my.domain.com" or "Doc3.my.domain.com".* This happens when the host name of the machine (with or without the domain information) is passed as the argument of the `licenseserver` CLI command that is used to register the server product with LicenseServer. For example: `<AltovaServerProduct> licenseserver Doc3`. This produces an FQDN such as: `Doc3.my.domain.com`.

  An FQDN is also produced when `localhost` is supplied on Windows 7 systems as the host name.

- *Host name without domain name. For example: "win80-x64_1" or "Doc3".* This happens on Windows 8 systems when `localhost` is given as the machine name.

- *localhost*. In some cases, `localhost` is also displayed as a machine name.

Note: If, during installation of the Altova server product on Windows machines, the machine is automatically registered with LicenseServer, `localhost` is used by the installer as the machine name.

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 with LicenseServer, go to the product's Setup page or its CLI and register it. *See:* Register FlowForce Server, Register MapForce Server, Register StyleVision Server, and Register RaptorXML(+XBRL) Server.

For more information, see the section, Assigning Licenses to Registered Products.

11.7.3 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.

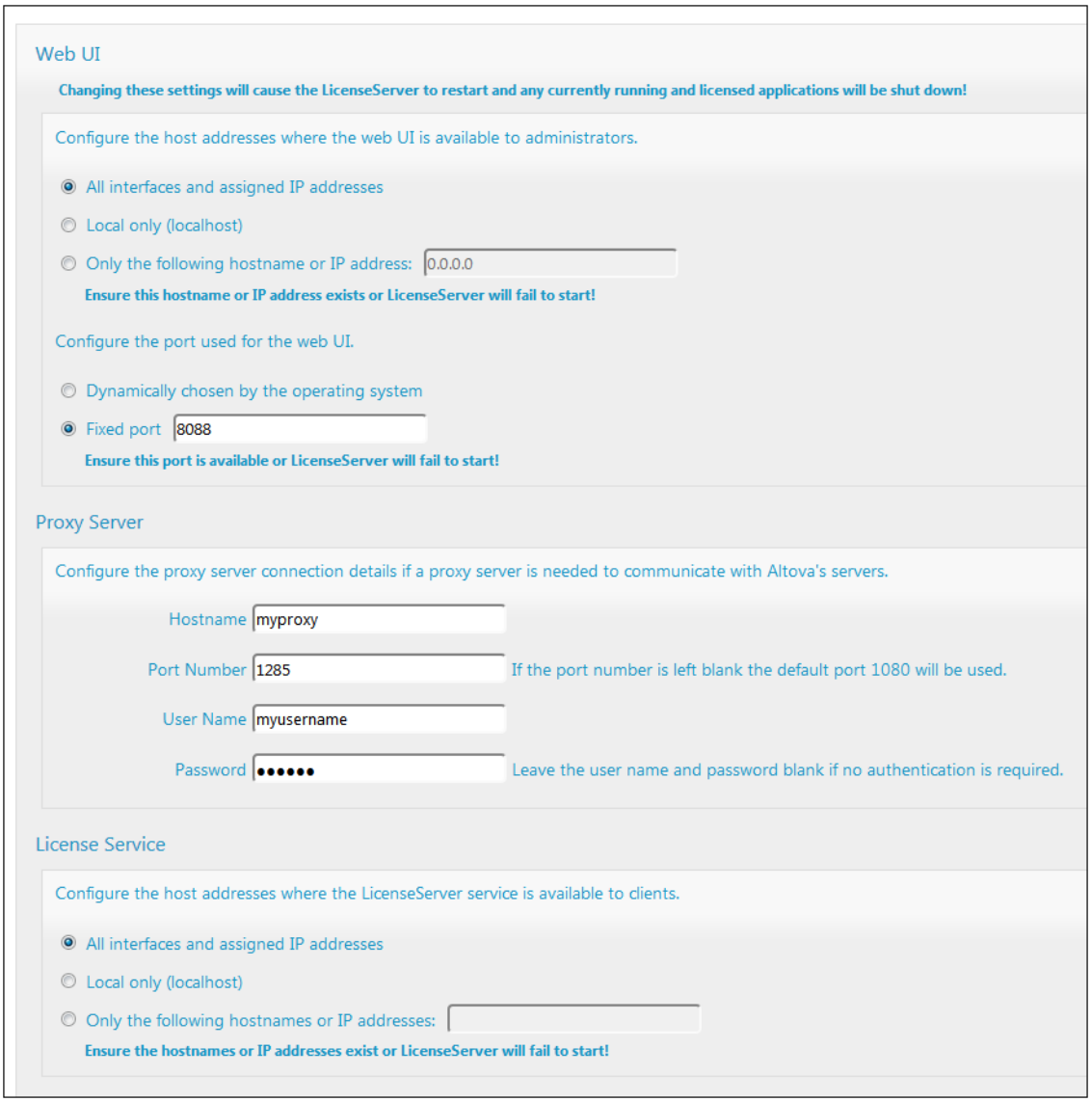
11.7.4 Settings

This section:

The **Settings** tab is described below. You can set the following:

- The password for logging in to LicenseServer. Enter the desired password and click **Change Password**.
- Test connectivity to Altova by clicking **Test Connection to Altova**. Note that you must save new settings (by clicking the **Save** button at the bottom of the pane) before testing the connection. The **Test Connection to Altova** button is disabled while the test is in progress, and becomes enabled again when the test has been completed.
- Client statistics
- Network settings for the web-based configuration page (Web UI), the proxy server used to connect to the Internet (if any), for and for LicenseServer (License Service). These settings are described in Network settings below.
- Email server settings and the alert mail recipient to contact in the event of a significant LicenseServer occurrence. These settings are described in Alert Mail settings below.
- After you change a setting, click **Save** at the bottom of the pane. A changed setting will not take effect till it is saved.

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**.
- *Proxy Server (available from v1.3 onwards):* If a proxy server is being used to connect to the Internet, the details of the proxy server must be entered in the Proxy Server pane (*see screenshot above*). These fields need to be filled in only if a proxy server is being used. Also, proxy servers often do not need authentication (a user-name and password), in which case these two fields can be left blank. To configure LicenseServer for the proxy server, enter the proxy server's host name, and, if required, a port number.
- *License Service:* IP addresses can vary from all interfaces and IP addresses on that machine to a fixed address. If you list hostnames and/or IP addresses, use a comma-separated list without any spaces (for example: `hostname1,IPAddress1,hostname2`).The port number is fixed at **35355**.

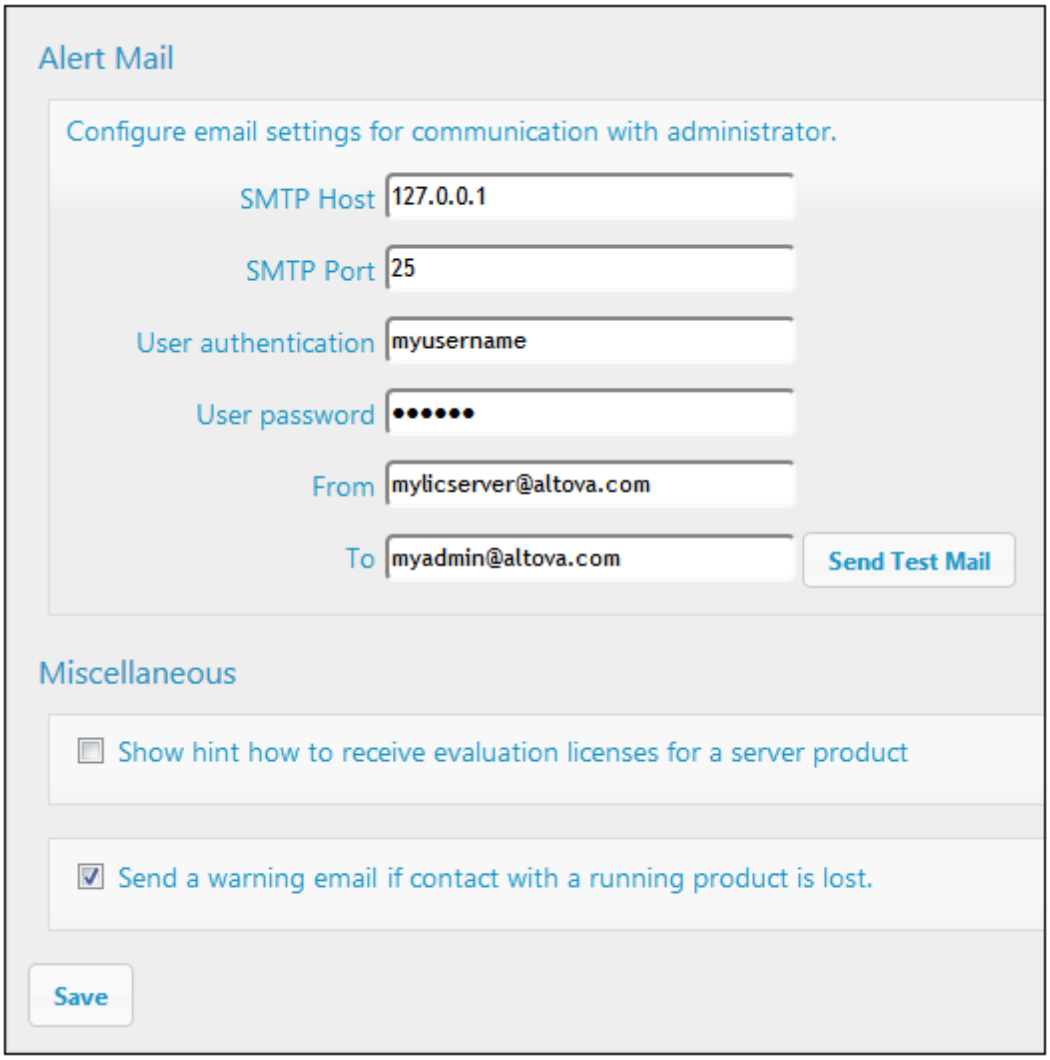
By default, these settings allow unrestricted access to LicenseServer and its configuration page from within the networks to which LicenseServer is connected. If you wish to restrict access to

either LicenseServer or its configuration page, enter the appropriate settings and click **Save**.

Run a connectivity test (*see above*) to check that the settings are correct.

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 the administrator that a connection is broken, an alert mail can be sent to an email address. The Alert Mail pane (*see screenshot below*) is where you enter settings 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. Note that such events are also recorded in the Messages tab, and can be looked up there.

Miscellaneous settings

Show hints for receiving and deploying evaluation licenses
Checking this box (*see secreenshot above*) displays, at the top of the configuration page, brief instructions about how to evaluate and deploy evaluation licenses.

Send a warning email if contact with a running product is lost
A warning message is sent from the *From* address to the *To* address if a connection with a product that is licensed and running is lost.

11.7.5 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