Altova RaptorXML Server 2024

User & Reference Manual

Altova RaptorXML Server 2024
User & Reference Manual

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

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

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

Published: 2023

© 2017-2023 Altova GmbH

Table of Contents

1 Introduction 9

2 About RaptorXML Server 10

2.1 Editions and Interfaces 11
2.2 System Requirements 15
2.3 Features 16
2.4 Supported Specifications 18
2.5 Notable Changes 20

3 Setting Up RaptorXML Server 21

3.1 Setup on Windows 22
3.1.1 Installing on Windows 22
3.1.2 Installing on Windows Server Core 23
3.1.3 Installing LicenseServer (Windows) 26
3.1.4 Network and Service Configuration (Windows) 27
3.1.5 Licensing RaptorXML Server (Windows) 28
3.2 Setup on Linux 33
3.2.1 Installing on Linux 33
3.2.2 Installing LicenseServer (Linux) 35
3.2.3 Licensing RaptorXML Server (Linux) 36
3.3 Setup on macOS 39
3.3.1 Installing on macOS 39
3.3.2 Installing LicenseServer (macOS) 40
3.3.3 Licensing RaptorXML Server (macOS) 41
3.4 Upgrading RaptorXML Server 44
3.5 Migrating RaptorXML Server to a New Machine 45

4 General Procedures 46

4.1 XML Catalogs ... 47
4.1.1 How Catalogs Work ... 47
4.1.2 Catalog Structure in RaptorXML Server ... 48
4.1.3 Customizing your Catalogs ... 50
4.1.4 Variables for Windows System Locations ... 52
4.2 Global Resources ... 53
4.3 Security Issues ... 55

5 Command Line Interface (CLI) 56

5.1 XML, DTD, XSD Validation Commands ... 58
5.1.1 valxml-withdtd (xml) ... 58
5.1.2 valxml-withxsd (xsi) ... 62
5.1.3 valdtd (dtd) ... 69
5.1.4 valxsd (xsd) ... 73
5.2 Well-formedness Check Commands ... 80
5.2.1 wfxml ... 80
5.2.2 wfdtd ... 84
5.2.3 wfany ... 87
5.3 XQuery Commands ... 92
5.3.1 xquery ... 92
5.3.2 xqueryupdate ... 100
5.3.3 valxquery ... 108
5.3.4 valxqueryupdate ... 114
5.4 XSLT Commands ... 121
5.4.1 xslt ... 121
5.4.2 valxslt ... 129
5.5 JSON/Avro Commands ... 136
5.5.1 avroextractschema ... 136
5.5.2 json2xml ... 139
5.5.3 jsonschema2xsd ... 144
5.5.4 valavro (avro) ... 148

5.5.5 valavrojson (avrojson) ... 152
5.5.6 valavroschema (avroschema) ... 156
5.5.7 valjsonschema (jsonschema) ... 159
5.5.8 valjson (json) ... 163
5.5.9 wfjson ... 168
5.5.10 xml2json ... 172
5.5.11 xsd2jsonschema ... 176
5.6 XML Signature Commands ... 183
5.6.1 xmlsignature-sign ... 183
5.6.2 xmlsignature-verify ... 187
5.6.3 xmlsignature-update ... 190
5.6.4 xmlsignature-remove ... 193
5.7 General Commands ... 195
5.7.1 valany ... 195
5.7.2 script ... 196
5.7.3 help ... 197
5.8 Localization Commands ... 199
5.8.1 exportresourcestrings ... 199
5.8.2 setdeflang ... 201
5.9 License Commands ... 202
5.9.1 licenseserver ... 202
5.9.2 assignlicense (Windows only) ... 203
5.9.3 verifylicense (Windows only) ... 205
5.10 Administration Commands ... 207
5.10.1 install ... 208
5.10.2 uninstall ... 208
5.10.3 start ... 209
5.10.4 setdeflang ... 210
5.10.5 licenseserver ... 211
5.10.6 assignlicense (Windows only) ... 212
5.10.7 verifylicense (Windows only) ... 214
5.10.8 createconfig ... 215
5.10.9 exportresourcestrings ... 216
5.10.10 debug ... 217
5.10.11 help ... 219

5.10.12 version ... 220
5.11 Options ... 221
5.11.1 Catalogs, Global Resources, ZIP Files ... 221
5.11.2 Messages, Errors, Help, Timeout, Version ... 222
5.11.3 Processing ... 223
5.11.4 XML ... 224
5.11.5 XSD ... 225
5.11.6 XQuery ... 227
5.11.7 XSLT ... 229
5.11.8 JSON/Avro ... 231
5.11.9 XML Signatures ... 232

6 Server APIs: HTTP REST, COM/.NET, Java 236

6.1 HTTP REST Client Interface ... 238
6.1.1 Server Setup ... 239
6.1.2 Client Requests ... 250
6.1.3 C# Example for REST API ... 270
6.2 COM/.NET API ... 274
6.2.1 COM Interface ... 274
6.2.2 COM Example: VBScript ... 274
6.2.3 .NET Interface ... 276
6.2.4 .NET Example: C# ... 277
6.2.5 .NET Example: Visual Basic .NET ... 280
6.3 Java API ... 283
6.3.1 Overview of the Interface ... 283
6.3.2 Example Java Project ... 284
6.4 Server API Reference ... 286
6.4.1 Interfaces/Classes ... 286
6.4.2 Enumerations ... 338

7 Engine APIs: Python and .NET 349

7.1 Licensing ... 351
7.2 Python API ... 352

7.2.1 Python API Versions ... 353
7.2.2 RaptorXML Server as a Python Package ... 355
7.2.3 Debugging Server-Side Python Scripts ... 358
7.2.4 Debugging Python Scripts in Visual Studio Code ... 358
7.2.5 FAQs ... 360
7.3 .NET Framework API ... 362

8 Schema Manager 363
8.1 Run Schema Manager ... 367
8.2 Status Categories ... 370
8.3 Patch or Install a Schema ... 372
8.4 Uninstall a Schema, Reset ... 374
8.5 Command Line Interface (CLI) ... 375
8.5.1 help ... 375
8.5.2 info ... 376
8.5.3 initialize ... 376
8.5.4 install ... 377
8.5.5 list ... 377
8.5.6 reset ... 378
8.5.7 uninstall ... 379
8.5.8 update ... 380
8.5.9 upgrade ... 380

9 Additional Information 381
9.1 Exit Codes ... 382
9.2 Schema Location Hints ... 383

10 Engine Information 384
10.1 XSLT and XQuery Engine Information ... 385
10.1.1 XSLT 1.0 ... 385
10.1.2 XSLT 2.0 ... 385
10.1.3 XSLT 3.0 ... 387
10.1.4 XQuery 1.0 ... 388

10.1.5 XQuery 3.1 ... 391
10.2 XSLT and XPath/XQuery Functions ... 393
10.2.1 Altova Extension Functions ... 394
10.2.2 Miscellaneous Extension Functions ... 485

Index **503**

1 Introduction

Altova RaptorXML 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.

This documentation

This documentation is delivered with the application and is also available online at the Altova website. This documentation is organized into the following sections:

- About RaptorXML 10
- Setting Up RaptorXML 46
- Command Line Interface 56
- Server APIs: HTTP, COM/.NET, Java 236
- Engine APIs: Python and .NET 349
- Additional Information 381
- Engine Information 384

Altova website: XML validation server, XML validator

Last updated: 16 October 2023

2 About 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 [11]. RaptorXML is available for Windows, Linux, and macOS. For more details of system support, see the section System Requirements [15].

Features and supported specifications

RaptorXML provides XML validation, XSLT transformations, and XQuery executions, each with a wide range of powerful options. See the section Features [16] for a broad list of available functionality and key features. The section Supported Specifications [18] provides a detailed list of the specifications to which RaptorXML conforms. For more information, visit the RaptorXML page at the Altova website.

© 2017-2023 Altova GmbH

2.1 Editions and Interfaces

Editions

RaptorXML is available in the following editions:

- *RaptorXML Server,* which is a fast server-based XML processing engine for the validation and processing of XML, XML Schema, XML Signature, XSLT, and XQuery documents.
- *RaptorXML+XBRL Server,* which provides all the functionality of RaptorXML Server plus a wide range of XBRL processing functionality.

See here[18] for a list of the supported specifications[18].

Interfaces

After you install RaptorXML, you can access it in one or more of the following ways:

- *Command Line Interface (CLI):* available for Windows, Linux, and macOS installations of RaptorXML
- *HTTP REST client interface:* uses RaptorXML's HTTP interface
- *COM/.NET server interface (Windows):* uses RaptorXML's (i) COM/.NET API and (ii) HTTP REST interface
- *Java server interface (Windows, Linux, macOS):* uses RaptorXML's (i) Java API and (ii) HTTP REST interface
- *Altova XMLSpy interface:* RaptorXML can be accessed from within the Altova XMLSpy user interface
- *Python engine interface:* uses (i) a RaptorXML Python-wheel in your Python environment and (ii) the Python API of RaptorXML in your Python script. In this way, RaptorXML functionality can be used in Python scripts together with third-party Python packages
- *.NET engine interface (Windows):* uses (i) a RaptorXML DLL and (ii) the .NET API of RaptorXML to create independent .NET applications that use RaptorXML functionality

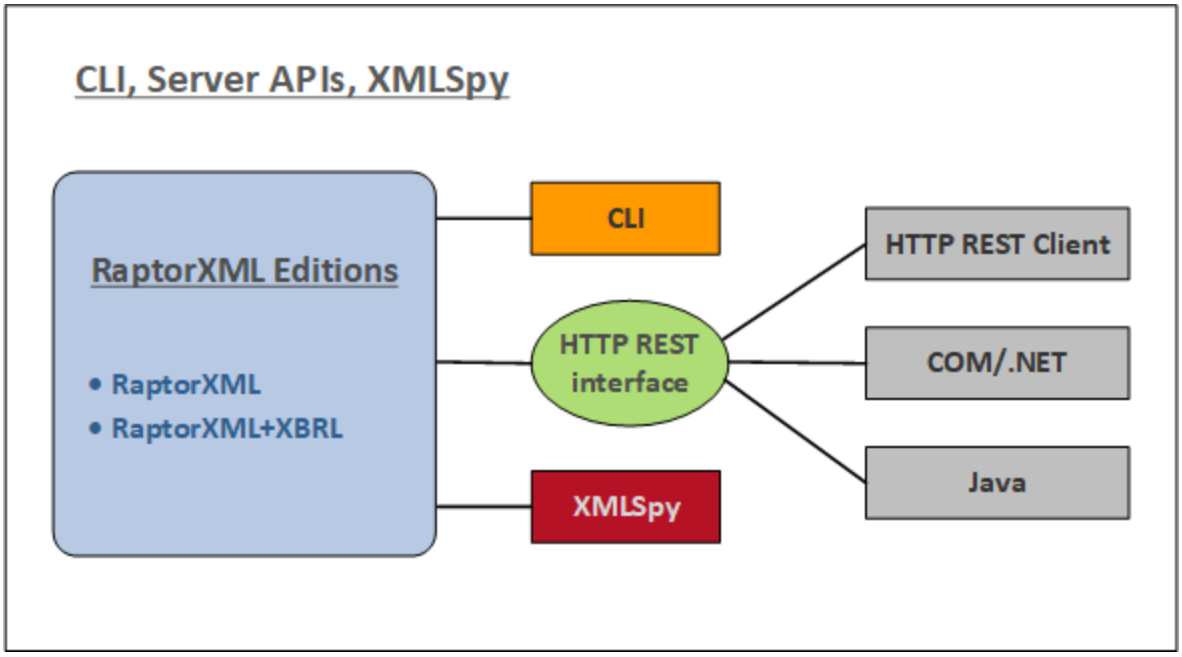
These seven interfaces can be organized into four groups:

- Command Line Interface (CLI)[56]
- Server APIs: HTTP, COM/.NET, Java[236]
- Engine APIs: Python and .NET[349]
- Altova XMLSpy[13]

CLI, Server APIs, and Altova XMLSpy

Access via the CLI, the Server APIs, and Altova XMLSpy can be visualized as in the figure below.

RaptorXML Server defines an HTTP REST interface, which is used by clients to dispatch validation jobs to the server. Clients can either access the HTTP REST interface directly or use the high-level COM/.NET and Java Server APIs. These APIs provide easy to use COM/.NET and Java classes which manage the creation and dispatch of the HTTP REST requests. Additionally, Altova XMLSpy can be configured to run validation jobs on a remote RaptorXML Server.



Command line interface (CLI)

- RaptorXML is licensed on the machine on which it is installed and this instance is accessed via the command line
- Can be installed on Windows, Linux, and macOS
- Provides command line usage [56] for validation and processing of XML, XML Schema, XML Signature, XQuery, and XSLT documents
- Python 3.11.5 is bundled in RaptorXML and will be used when a Python script is invoked with the `--script` option

HTTP REST client interface

- RaptorXML is licensed on the machine on which it is installed and this instance is accessed via an HTTP REST client interface [238]
- 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.
- Python 3.11.5 is bundled in RaptorXML and will be used when a Python script is invoked with the `--script` option

COM/.NET interface

- Available on Windows only
- RaptorXML is automatically registered as a COM server object when installed, and so can be invoked from within applications and scripting languages that have programming support for COM calls
- RaptorXML is licensed on the machine on which it is installed
- The .NET interface is built as a wrapper around the COM interface

 © 2017-2023 Altova GmbH

- The COM/.NET Server API[274] of RaptorXML provides objects that can be used in COM/.NET scripting languages to access RaptorXML functionality
- Python 3.11.5 is bundled in RaptorXML and will be used when a Python script is invoked with the `--script` option

Java interface

- RaptorXML is licensed on the machine on which it is installed and this instance is accessed via a Java program
- RaptorXML functionality is available in the Java Server API[283] as Java classes that can be used in Java programs.
- Python 3.11.5 is bundled in RaptorXML and will be used when a Python script is invoked with the `--script` option

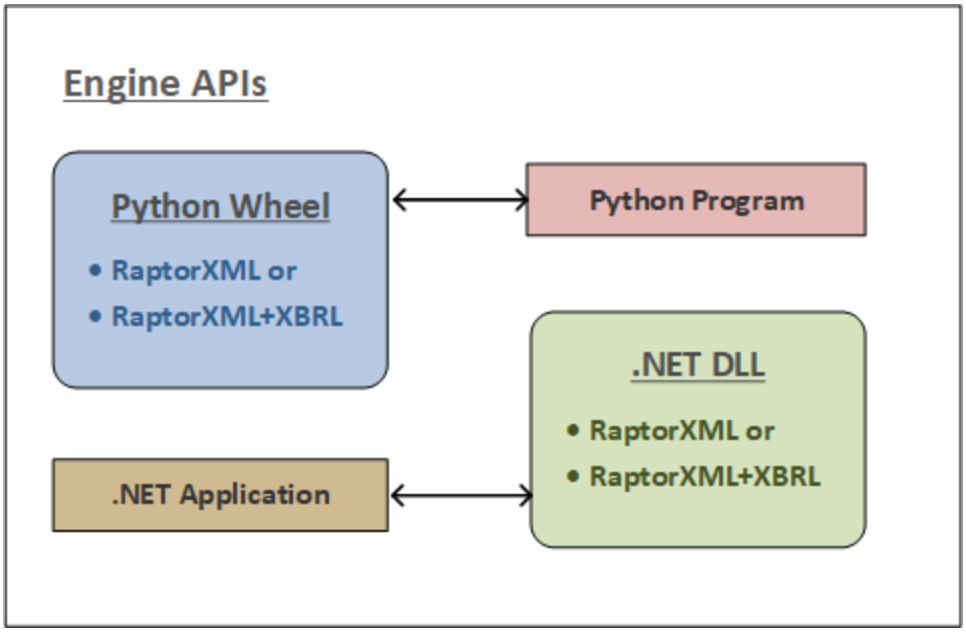
Altova XMLSpy

- If you have installed and licensed Altova XMLSpy and if XMLSpy can access RaptorXML Server across a network, then you can use RaptorXML Server from withing the XMLSpy GUI to validate XML documents, as well as run XSLT and XQuery transformations.
- You can validate the active document or all the documents in an XMLSpy project folder.
- The validation results are displayed in the Messages window of the XMLSpy GUI.
- In XMLSpy, you can (i) validate documents or (ii) run XSLT/XQuery transformation by using either XMLSpy's engines or RaptorXML Server.
- One of the main advantages of using Raptor is that you can configure individual validations by means of a large range of validation options. Furthermore, you can store a set of Raptor options as a "configuration" in XMLSpy, and then select one of your defined configurations for a particular Raptor validation. Using Raptor is also advantageous when large data collections are to be validated.

Engine APIs

The Engine APIs[349] are different than the Server APIs in that RaptorXML is contained in the Python wheel and in the .NET DLL that are used, respectively, by Python programs and .NET applications (*see figure below*). These programs/applications must use, respectively, Raptor's Python API[352] and Raptor's .NET API[362] in order to access RaptorXML functionality.

Note: The functionality provided by the Python API[352] and .NET API[362] are considerably greater than that provided by either the CLI or the Server APIs; for example, the ability to read documents and manipulate data.



Python interface

- RaptorXML is available in a Python wheel package that can be installed in your Python 3.11.5 environment
- A Python program can then be written that uses objects from RaptorXML's Python API[352]. This API provides much more functionality than is available in the CLI, and it can be combined with the functionality provided by third-party libraries in your Python environment
- When RaptorXML functionality is called via RaptorXML's Python wheel, a check is carried out for a valid RaptorXML license on that machine before the command is executed

.NET interface

- RaptorXML is available in a DLL that can be embedded in an application that supports the .NET Framework. See the section .NET Framework API[362] for information about the API.
- RaptorXML's .NET API[362] provides access to RaptorXML. The available functionality is much more than that which is available in the RaptorXML CLI.
- When RaptorXML functionality is called via a .NET application, a check is carried out for a valid RaptorXML license on that machine

 © 2017-2023 Altova GmbH

2.2 System Requirements

RaptorXML Server is supported on the following operating systems:

- *Windows*

  Windows 10, Windows 11

- *Windows Server*

  Windows Server 2012 or newer

System Requirements (Linux)

- Red Hat Enterprise Linux 7 or newer
- CentOS 7, CentOS Stream 8
- Debian 9 or newer
- Ubuntu 18.04, 20.04, 22.04
- AlmaLinux 9.0
- Rocky Linux 9.0

Prerequisites

- Perform installation either as **root** user or as a user with **sudo** privileges.
- The previous version of RaptorXML Server must be uninstalled before a new one is installed.
- If you plan to use Altova's Charts functionality, then at least one font must be installed on your system to ensure that charts will be rendered correctly. To list installed fonts, use, for example, the **`fc-list`** command of the Fontconfig library.
- The following libraries are required as a prerequisite to install and run the application. If the packages below are not already available on your Linux machine, run the command `yum` (or `apt-get` if applicable) to install them.

Required by	CentOS, RedHat	Debian	Ubuntu
LicenseServer	libidn or libidn2, krb5-libs, fontconfig	libidn2-0, libgssapi-krb5-2, fontconfig	libidn2-0, libgssapi-krb5-2, libfontconfig
RaptorXML Server	libidn or libidn2, krb5-libs	libidn2-0, libgssapi-krb5-2	libidn2-0, libgssapi-krb5-2

- *macOS*

  macOS 11 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.

2.3 Features

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

XML Validation

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

XSLT Transformations

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

XQuery Execution

- Executes XQuery 1.0 and 3.0 documents
- XQuery and XML documents can be provided as a file (via a URL) or, in the case of COM usage, as a text string
- Output is returned as a file (at a named location) or, in the case of COM usage, as a text string
- External XQuery variables can be supplied via the command line and via the COM interface
- Serialization options include: output encoding, output method (that is, whether the output is XML, XHTML, HTML, or text), omitting the XML declaration, and indentation

JSON and Avro Validation/Conversion

- Validation of JSON schema and Avro schema documents
- Validation of JSON instances against JSON schemas and Avro schemas
- Validation of Avro binaries
- Conversion of Avro binaries to Avro schema and Avro data in JSON format
- Conversion of Avro JSON data to Avro binary

Hyper-performance Features

- Ultra-high performance code optimizations
  - Native instruction-set implementations
  - 32-bit or 64-bit version

© 2017-2023 Altova GmbH

- 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
- RaptorXML functionality in a Python package enables import of the functionality as a Python library
- .NET Framework API allows access to underlying XML data model
- COM API on Windows platform
- Java API everywhere
- XPath Extension functions Java, .NET, and more
- Streaming serialization
- Built-in HTTP server with REST validation API

For more information, see the section Supported Specifications 18 and the Altova website.

2.4 Supported Specifications

RaptorXML supports the specifications listed below.

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)
- XSLT 2.0 and XQuery 1.0 Serialization (Second Edition)
- XML Path Language (XPath) 3.0
- XML Path Language (XPath) 3.1
- XQuery 3.0: An XML Query Language
- XQuery Update Facility 1.0
- XPath and XQuery Functions and Operators 3.0
- XSLT and XQuery Serialization 3.0

W3C Working Drafts & Candidate Recommendations

Website: World Wide Web Consortium (W3C)

- XSL Transformations (XSLT) Version 3.0 (subset)
- XQuery 3.1: An XML Query Language
- XPath and XQuery Functions and Operators 3.1
- XQuery Update Facility 3.0
- XSLT and XQuery Serialization 3.1

OASIS Standards

Website: OASIS Standards

© 2017-2023 Altova GmbH

- XML Catalogs V 1.1 - OASIS Standard V1.1

JSON/Avro Standards

Websites: JSON Schema and Apache Avro

- JSON Schema Draft 4
- JSON Schema Draft 6
- JSON Schema Draft 7
- JSON Schema Draft 2019-09
- JSON Schema Draft 2020-12
- Apache Avro™ 1.8.1

2.5 Notable Changes

Given below are changes in each version that might need your attention.

v2024

On the command line, the **`--network-timeout`** option takes a value in milliseconds from this release onwards (instead of in seconds as was the case in previous releases). The option can be set for a number of commands and, in the description of the command, is listed under *Common Options*. For an example, see the valxml-withxsd (xsi) [62] command.

© 2017-2023 Altova GmbH

3 Setting Up RaptorXML Server

This section describes installation, licensing and other setup procedures. It is organized into the following sections:

- Setup on Windows 22
- Setup on Linux 33
- Setup on macOS 39
- Upgrading RaptorXML Server 44
- Migrating RaptorXML Server to a New Machine 45

3.1 Setup on Windows

This section describes the installation[22] and licensing[28] of RaptorXML Server on Windows systems.

System requirements (Windows)

- Windows 10, Windows 11
- Windows Server 2012 or newer

Prerequisites

- Perform installation as a user with administrative privileges.
- From version 2021 onwards, a 32-bit version of RaptorXML Server cannot be installed over a 64-bit version, or a 64-bit version over a 32-bit version. You must either (i) remove the older version before installing the newer version or (ii) upgrade to a newer version that is the same bit version as your older installation.

3.1.1 Installing on Windows

RaptorXML Server is available for installation on Windows systems. The broad installation and setup procedure is described below. For detailed information about specific parts of the installation procedure, see their respective topics.

Installing RaptorXML Server

RaptorXML Server can be installed on Windows systems as follows:

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

The installers of both RaptorXML Server and FlowForce Server are available at the Altova Download Center (http://www.altova.com/download.html). You can select your installation language from the box in the lower left area of the wizard. Note that this selection also sets the default language of RaptorXML Server. You can change the language later from the command line.

After installation, the RaptorXML Server executable will be located by default at:

```
<ProgramFilesFolder>\Altova\RaptorXMLServer2024\bin\RaptorXML.exe
```

All the necessary registrations to use RaptorXML 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 Server executable as a COM server object and adding the `Altova.RaptorXML.dll` file to the .NET reference library.

 © 2017-2023 Altova GmbH

Installing on Windows Server Core

Windows Server Core has no GUI and must be installed via the command line. See the section Installing on Windows Server Core 23 for information about how to do this.

Uninstalling RaptorXML Server

Uninstall RaptorXML Server as follows:

1. Right-click the Windows **Start** button and select **Settings**.
2. Open the Control Panel (start typing "Control Panel" and click the suggested entry).
3. Under *Programs*, click **Uninstall a program**.
4. In Control Panel, select RaptorXML Server and click **Uninstall**.

Evaluation license

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

3.1.2 Installing on Windows Server Core

Windows Server Core is a minimal Windows installation that does not use a number of GUI features. You can install RaptorXML Server on a Windows Server Core machine as follows:

1. Download the RaptorXML Server installer executable from the Altova website. This file is named **`RaptorXMLServer<version>.exe`**. Make sure to choose the executable matching your server platform (32-bit or 64-bit).
2. On a standard Windows machine (not the Windows Server Core machine), run the command **`RaptorXMLServer<version>.exe /u`**. This unpacks the **`.msi`** file to the same folder as the installer executable.
3. Copy the unpacked **`.msi`** file to the Windows Server Core machine.
4. If you are updating an earlier version of RaptorXML Server, shut down RaptorXML Server before carrying out the next step.
5. Use the **`.msi`** file for the installation by running the command **`msiexec /i RaptorXMLServer.msi`**. This starts the installation on Windows Server Core.

Note: When upgrading to a major version, you can retain your RaptorXML Server settings by using the properties listed in the subsections of this section: (i) Webserver Properties 25, (ii) SSL-Webserver Properties 25, and (iii) Service Properties 26.

Important: Keep the MSI file!

Note the following points:

- Keep the extracted **`.msi`** file in a safe place. You will need it later to uninstall, repair, or modify your installation.
- If you want to rename the MSI file, do this before you install RaptorXML Server.

- The MSI filename is stored in the registry. You can update its name there if the filename has changed.

Register RaptorXML Server with LiceseServer

If you are installing RaptorXML Server for the first time or are upgrading to a **major version**, you will need to register RaptorXML Server with an Altova LicenseServer on your network. If you are upgrading to a non-major version of RaptorXML Server, then the previous LicenseServer registration will be known to the installation and there is no need to register RaptorXML Server with LicenseServer. However, if you want to change the LicenseServer that is used by RaptorXML Server at any time, then you will need to register RaptorXML Server with the new LicenseServer.

To register RaptorXML Server with an Altova LicenseServer during installation, run the installation command with the **REGISTER_WITH-LICENSE_SERVER** property, as listed below, providing the name or address of the LicenseServer machine as the value of the property, for example:

```
msiexec /i RaptorXMLServer.msi REGISTER_WITH_LICENSE_SERVER="localhost"
```

To register RaptorXML Server with an Altova LicenseServer after installation, run the following command:

```
msiexec /r RaptorXMLServer.msi REGISTER_WITH_LICENSE_SERVER="<MyLS-IPAddress>"
```

Useful commands

Given below are a set of commands that are useful in the installation context.

To test the return value of the installation, run a script similar to that below. The return code will be in the `%errorlevel%` environment variable. A return code of `0` indicates success.

```
start /wait msiexec /i RaptorXMLServer.msi /q
echo %errorlevel%
```

For a silent installation with a return code and a log of the installation process:

```
start /wait msiexec /i RaptorXMLServer.msi /q /L*v! <pathToInstallLogFile>
```

To modify the installation:

```
msiexec /m RaptorXMLServer.msi
```

To repair the installation:

```
msiexec /r RaptorXMLServer.msi
```

To uninstall RaptorXML Server:

```
msiexec /x RaptorXMLServer.msi
```

To uninstall RaptorXML Server silently and report the detailed outcome in a log file:

```
start /wait msiexec /x RaptorXMLServer.msi /q /L*v! <pathToUninstallLogFile>
```

To install RaptorXML Server using another langauge (available language codes are: German=`de`; Spanish=`es`; French=`fr`):

```
msiexec /i RaptorXMLServer.msi INSTALLER_LANGUAGE=<languageCode>
```

Note: On Windows Server Core, the charts functionality of RaptorXML Server will not be available.

 © 2017-2023 Altova GmbH

3.1.2.1 Webserver Properties

You can configure the RaptorXML Server web server by using the properties given below. To set a property, run the installation command with the property setting appended, like this:

```
msiexec /i RaptorXMLServer.msi RXML_WebServer_Host=127.0.0.1
```

List of properties

Properties of the RaptorXML Server web server:

RXML_WebServer_Host=<IP4 Address>
Use `127.0.0.1` if you want to access the web server from this machine only. Use `0.0.0.0` to make the web server accessible globally.

RXML_WebServer_Port=<Port Number>
Specifies the port that is used to access the web server.

RXML_WebServer_Enabled=<0 or 1>
Select `1` to enable listening at the currently set port. Select `0` to disable listening at this port.

3.1.2.2 SSL-Webserver Properties

You can configure the RaptorXML Server SSL web server by using the properties given below. To set a property, run the installation command with the property setting appended, like this:

```
msiexec /i RaptorXMLServer.msi RXML_SSLWebServer_Host=127.0.0.1
```

List of properties

To configure the RaptorXML Server SSL web server, use the following properties:

RXML_SSLWebServer_Host=<IP4 Address>
Use `127.0.0.1` if you want to access the SSL web server (for encrypted transmission) from this machine only. Use `0.0.0.0` to make the SSL web server accessible globally.

RXML_SSLWebServer_Port=<Port Number>
Specifies the port that is used to access the SSL web server (for encrypted transmission).

RXML_SSLWebServer_Enabled=<0 or 1>
Select `1` to enable listening at the currently set port. Select `0` to disable listening at this port.

RXML_SSLWebServer_Certificate=<Path-to-certificate-file>
Full path to a SSL certificate, enclosed in double-quotes.

RXML_SSLWebServer_PrivateKey=<Path-to-private-key-file>
Full path to a private key file, enclosed in double-quotes.

3.1.2.3 Service Properties

You can configure the RaptorXML Server service by using the properties given below. To set a property, run the installation command with the property setting appended, like this:

```
msiexec /i RaptorXMLServer.msi RXML_Service_DisplayName=RaptorXMLServer
```

List of properties

To configure RaptorXML Server services, use the following properties:

RXML_Service_DisplayName=<Serveice Display Name>
Name that will be displayed for the service. Enclose the name in double quotes.

RXML_Service_StartType=<Startup Type>
Specifies how the service is started during a system start-up. Values can be one of: **auto | auto-delayed | demand | disabled**.

RXML_Service_Username=<UserName>
Specifies the log-on user for the service. Use one of: **LocalSystem | NT Authority\LocalService | NT Authority\NetworkService | <any user with relevant rights>**.

RXML_Service_Password=<Password>
The password of the service's start user in plain text.(Hint: Use the installer's user interface to avoid entering plain text passwords.) No password is required if the user name is any of: **LocalSystem | NT Authority\LocalService | NT Authority\NetworkService**.

3.1.3 Installing LicenseServer (Windows)

In order for RaptorXML Server to work, it must be licensed via an Altova LicenseServer on your network. When you install RaptorXML Server or FlowForce Server on Windows systems, you can install LicenseServer together with RaptorXML Server or FlowForce Server. If a LicenseServer is already installed on your network, you do not need to install another one—unless a newer version of LicenseServer is required. (*See next point, LicenseServer versions*.)

During the installation process of RaptorXML Server or FlowForce Server, check or uncheck the option for installing LicenseServer as appropriate. Note the following points:

- If you have not installed LicenseServer yet, leave the default settings as is. The wizard will install the latest version on the computer where you are running the wizard.
- If you have not installed LicenseServer yet and want to install Altova LicenseServer on another computer, clear the check box *Install Altova LicenseServer on this machine* and choose **Register Later**. In this case, you will need to install LicenseServer separately and register RaptorXML Server afterwards.
- If LicenseServer has already been installed on your computer but is a lower version than the one indicated by the installation wizard, leave the default setting (for upgrading to the newer version) as is.

 © 2017-2023 Altova GmbH

In this case, the installation wizard will automatically upgrade your LicenseServer version. The existing registration and licensing information will be carried over to the new version of LicenseServer.

- If LicenseServer has already been installed on your computer or network and has the same version as the one indicated by the wizard, do the following:
  - Clear the check box *Install Altova LicenseServer on this machine*.
  - Under *Register this product with*, choose the LicenseServer with which you want to register RaptorXML Server. Alternatively, choose **Register Later**. Note that you can always select **Register Later** if you want to ignore the LicenseServer associations and carry on with the installation of RaptorXML Server.

For information about how to register and license RaptorXML Server with Altova LicenseServer, see the section Licensing RaptorXML Server[28].

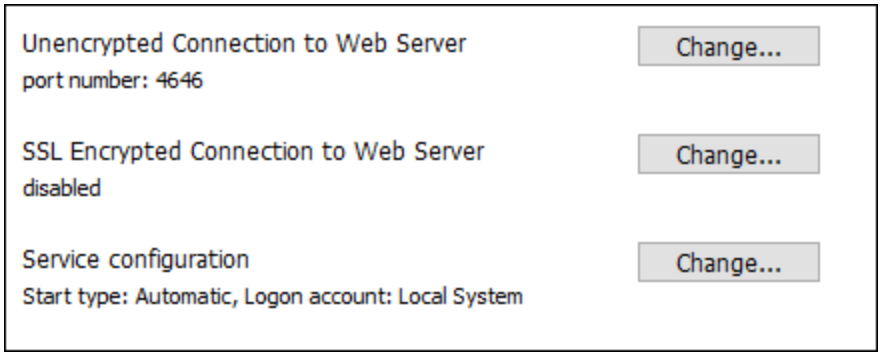
LicenseServer versions

- Altova server products must be licensed either with the version of LicenseServer that is appropriate to the installed RaptorXML Server version or with a later version of LicenseServer.
- The LicenseServer version that is appropriate for a particular version of RaptorXML Server is displayed during the installation of RaptorXML Server. You can install this version of LicenseServer along with RaptorXML Server. Alternatively, 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 Server.
- If you install a new version of RaptorXML Server and if your installed LicenseServer version is older than the appropriate LicenseServer, install the latest version of LicenseServer 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: 3.12

3.1.4 Network and Service Configuration (Windows)

During the installation of RaptorXML Server, you can configure settings for accessing RaptorXML Server via the network and for running RaptorXML Server as a Windows service (*screenshot below*).



The settings listed below are available. Leave the default settings as they are if they are acceptable to you or if you are not sure about them. If you wish to change a setting, select its **Change** button (*see screenshot above*).

- The port to use for unencrypted communication with RaptorXML Server.
- Whether secure (SSL-encrypted) connections to RaptorXML Server are allowed. If yes, then on which port. By default, secure connections are disabled. For more information, see the section about setting up SSL encryption 247.
- Windows service settings. These include:
  - The way RaptorXML Server should start as a Windows service: automatic, on demand, delayed automatic, or disabled.
  - The user account to be used by RaptorXML Server for the Windows service: *Local System, Local Service, Network Service,* or *Other User*. If you select *Other User*, you can set the username and password of this user, similar to how this is done in the Windows Services management console. Note that the selected user must have read/write access to `C:\ProgramData\Altova`. Otherwise, the installation or startup could fail.

You can change the settings after installation. To modify the Windows service configuration, open the Windows Services management console (by typing `Services.msc` in a command line window) and change the required service from there.

3.1.5 Licensing RaptorXML Server (Windows)

In order to use RaptorXML Server, it must be licensed with Altova LicenseServer. Licensing is a two-step process:

1. **Register RaptorXML Server** with LicenseServer. Registration is done from RaptorXML Server.
2. **Assign a license** to RaptorXML Server from LicenseServer. Download the latest version of LicenseServer from the Altova website, and install it on your local machine or a machine on your network.

These steps are described in this section. For detailed information, see the LicenseServer user manual at the Altova website.

 © 2017-2023 Altova GmbH

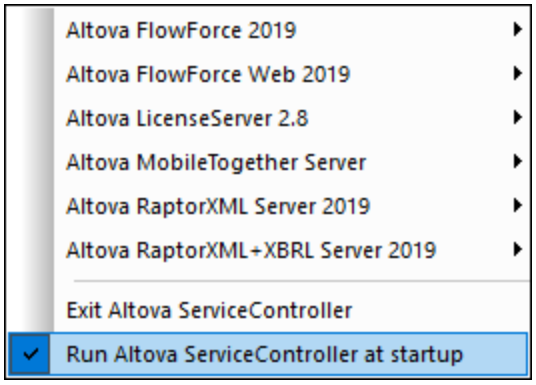
3.1.5.1 Start LicenseServer, RaptorXML Server

Altova LicenseServer (LicenseServer for short) and RaptorXML Server are both started via Altova ServiceController.

Altova ServiceController

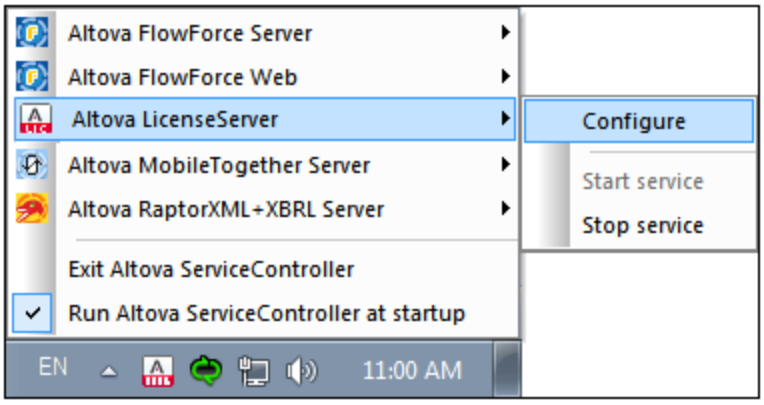
Altova ServiceController (ServiceController for short) is an application for conveniently starting, stopping and configuring Altova services **on Windows systems**. ServiceController is installed with Altova LicenseServer and with Altova server products that are installed as services (DiffDog Server, FlowForce Server, Mobile Together Server, and RaptorXML(+XBRL) Server). ServiceController 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**.



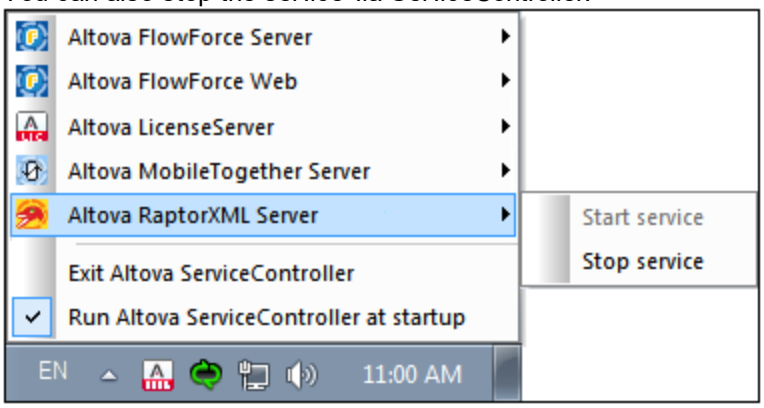
Start LicenseServer

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, then the *Start Service* option will be disabled. You can also stop the service via ServiceController.



Start RaptorXML Server

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



Note: If RaptorXML Server has been licensed to run only single-thread executions (typically because your machine is multiple-core, but your license is single-core), then you can use only one instance of RaptorXML Server at a time: either as a service or from the command line. This is because the single-core license will be assigned automatically to the first instance that is started and is currently running. The second instance cannot be started until the first instance stops running.

- If you wish to use RaptorXML Server from the command line, but the service is already running, you must stop the service before using the command line.
- If you wish to start RaptorXML Server as a service, make sure that no command line action is currently being executed. Otherwise, you will not be able to start the service.

 © 2017-2023 Altova GmbH

3.1.5.2 Register RaptorXML Server

In order to be able to license RaptorXML Server from Altova LicenseServer, RaptorXML Server must be registered with LicenseServer

To register RaptorXML Server from the command line interface, use the `licenseserver` command and supply the address of the LicenseServer machine.

```
RaptorXML licenseserver [options] ServerName-Or-IP-Address
```

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

```
RaptorXML licenseserver localhost
```

If RaptorXML Server was installed as part of a FlowForce Server installation, registering FlowForce Server with LicenseServer will automatically also register RaptorXML Server. 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**. For more information, see Register FlowForce Server.

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

Note: For more information about registering Altova products with LicenseServer, see the LicenseServer user manual.

3.1.5.3 License RaptorXML Server

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

The licensing of Altova server products is based on the number of processor cores available on the product machine. For example, a dual-core processor has two cores, a quad-core processor four cores, a hexa-core processor six cores, and so on. The number of cores licensed for a product must be greater than or equal to the number of cores available on that server machine, whether the server is a physical or virtual machine. For example, if a server has eight cores (an octa-core processor), you must purchase at least one 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 one 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, would have less processing speed than if all available cores on the server were utilized.

Note: Each Altova server product license can be used for only one client machine at a time, even if the license has unused licensing capacity. (A client machine is the machine on which the Altova server product is installed.) For example, if a 10-core license is used for a client machine that has 6 CPU cores, then the remaining 4 cores of licensing capacity cannot be used simultaneously for another client machine.

Single-thread execution
If an Altova server product allows single-thread execution, an option for *Single-thread execution* will be available. In these cases, if an Altova server-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, because 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 in LicenseServer, select the *Limit to single thread execution* check box for that product.

Estimate of core requirements
There are various external factors that influence the data volumes and processing times your server can handle (for example: the hardware, the current load on the CPU, and memory allocation of other applications running on the server). In order to measure performance as accurately as possible, test the applications in your environment with data volumes and in conditions that approximate as closely as possible to real business situations.

© 2017-2023 Altova GmbH

3.2 Setup on Linux

This section describes the installation[33] and licensing[36] of RaptorXML Server on Linux systems (Debian, Ubuntu, CentOS, RedHat).

System Requirements (Linux)

- Red Hat Enterprise Linux 7 or newer
- CentOS 7, CentOS Stream 8
- Debian 9 or newer
- Ubuntu 18.04, 20.04, 22.04
- AlmaLinux 9.0
- Rocky Linux 9.0

Prerequisites

- Perform installation either as **root** user or as a user with **sudo** privileges.
- The previous version of RaptorXML Server must be uninstalled before a new one is installed.
- If you plan to use Altova's Charts functionality, then at least one font must be installed on your system to ensure that charts will be rendered correctly. To list installed fonts, use, for example, the **`fc-list`** command of the Fontconfig library.
- The following libraries are required as a prerequisite to install and run the application. If the packages below are not already available on your Linux machine, run the command `yum` (or `apt-get` if applicable) to install them.

Required by	CentOS, RedHat	Debian	Ubuntu
LicenseServer	libidn or libidn2, krb5-libs, fontconfig	libidn2-0, libgssapi-krb5-2, fontconfig	libidn2-0, libgssapi-krb5-2, libfontconfig
RaptorXML Server	libidn or libidn2, krb5-libs	libidn2-0, libgssapi-krb5-2	libidn2-0, libgssapi-krb5-2

3.2.1 Installing on Linux

RaptorXML Server is available for installation on Linux systems. Its installation and setup procedure is described below. Perform installation either as **root** user or as a user with **sudo** privileges.

Integration with FlowForce

If you are installing RaptorXML Server together with FlowForce Server, it is recommended that you install FlowForce Server first. If you install RaptorXML Server before FlowForce Server, then, after having installed both RaptorXML Server and FlowForce Server, run the following command:

```
cp /opt/Altova/RaptorXMLServer2024/etc/*.tool /opt/Altova/FlowForceServer2024/tools
```

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

Uninstall RaptorXML Server

If you need to uninstall a previous version of RaptorXML Server, 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 Server is not installed, go ahead with the installation as documented below in *Install RaptorXML Server*.

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

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

On Debian and Ubuntu systems, it might happen that RaptorXML Server still appears in the list of installed products after it has been uninstalled. In this case, run the **purge** command to clear RaptorXML Server from the list. You can also use the **purge** command *instead* of the **remove** command listed above.

```
[Debian, Ubuntu]:   sudo dpkg --purge raptorxmlserver
```

Download the RaptorXML Server Linux package

RaptorXML Server installation packages for the following Linux systems are available at the Altova website.

Distribution	Package extension
Debian	.deb
Ubuntu	.deb
CentOS	.rpm
RedHat	.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 Server, you may want to download LicenseServer from the Altova website at the same time as you download RaptorXML Server, rather than download it at a later time.

Install RaptorXML 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 Server with the following command:

```
[Debian]:   sudo dpkg --install raptorxml-2024-debian.deb
[Ubuntu]:   sudo dpkg --install raptorxml-2024-ubuntu.deb
[CentOS]:   sudo rpm -ivh raptorxml-2024-1.x86_64.rpm
```


© 2017-2023 Altova GmbH

```
[RedHat]:   sudo rpm -ivh raptorxml-2024-1.x86_64.rpm
```

Note: You may need to adjust the name of the package above to match the current release or service pack version.

The RaptorXML Server package will be installed in the folder:

```
/opt/Altova/RaptorXMLServer2024
```

3.2.2 Installing LicenseServer (Linux)

In order for RaptorXML Server to work, it 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 Server (*see previous topic* 33).

```
[Debian]:   sudo dpkg --install licenseserver-3.12-debian.deb
[Ubuntu]:   sudo dpkg --install licenseserver-3.12-ubuntu.deb
[CentOS]:   sudo rpm -ivh licenseserver-3.12-1.x86_64.rpm
[RedHat]:   sudo rpm -ivh licenseserver-3.12-1.x86_64.rpm
```

The LicenseServer package will be installed in:

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

For information about how to register and license RaptorXML Server with Altova LicenseServer, see the section Licensing RaptorXML Server 36. Also see the LicenseServer documentation for more detailed information.

LicenseServer versions

- Altova server products must be licensed either with the version of LicenseServer that is appropriate to the installed RaptorXML Server version or with a later version of LicenseServer.
- The LicenseServer version that is appropriate for a particular version of RaptorXML Server is displayed during the installation of RaptorXML Server. You can install this version of LicenseServer along with RaptorXML Server. Alternatively, 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 Server.
- If you install a new version of RaptorXML Server and if your installed LicenseServer version is older than the appropriate LicenseServer, install the latest version of LicenseServer 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: 3.12

3.2.3 Licensing RaptorXML Server (Linux)

In order to use RaptorXML Server, it must be licensed with Altova LicenseServer. Licensing is a two-step process:

1. **Register RaptorXML Server** with LicenseServer. Registration is done from RaptorXML Server.
2. **Assign a license** to RaptorXML Server from LicenseServer. Download the latest version of LicenseServer from the Altova website, and install it on your local machine or a machine on your network.

These steps are described in this section. For detailed information, see the LicenseServer user manual at the Altova website.

3.2.3.1 Start LicenseServer, RaptorXML Server

This topic describes how to start Altova LicenseServer (LicenseServer for short) and RaptorXML Server. To be able to start these programs, you can use one of the following options: (i) you can be the root user and leave out the `sudo` keyword from the commands listed below (leaving out `sudo` is optional), or (ii) you can run the `sudo` command as a normal user with the corresponding permissions for `sudo`.

Start LicenseServer

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

[≥ Debian 8], [≥ CentOS 7], [≥ Ubuntu 15]	**sudo systemctl start licenseserver**

If at any time you need to stop LicenseServer, replace **start** with **stop** in the command above. For example:

```
sudo systemctl stop licenseserver
```

Start RaptorXML Server

Start RaptorXML Server as a daemon with the command appropriate for your system from those listed below.

[≥ Debian 8], [≥ CentOS 7], [≥ Ubuntu 15]	**sudo systemctl start raptorxmlserver**

If at any time you need to stop RaptorXML Server, replace **start** with **stop** in the command above. For example:

```
sudo systemctl stop raptorxmlserver
```

Check status of daemons

To check if a daemon is running, run the following command, replacing **<servicename>** with the name of the daemon you want to check:

© 2017-2023 Altova GmbH

```
sudo service <servicename> status
```

3.2.3.2 Register RaptorXML Server

To register RaptorXML Server from the command line interface, use the `licenseserver` command:

```
sudo /opt/Altova/RaptorXMLServer2024/bin/raptorxml licenseserver [options] ServerName-Or-IP-Address
```

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

```
sudo /opt/Altova/RaptorXMLServer2024/bin/raptorxml 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 Server executable is:

```
/opt/Altova/RaptorXMLServer2024/bin/
```

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

Note: For more information about registering Altova products with LicenseServer, see the LicenseServer user manual.

3.2.3.3 License RaptorXML Server

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

The licensing of Altova server products is based on the number of processor cores available on the product machine. For example, a dual-core processor has two cores, a quad-core processor four cores, a hexa-core processor six cores, and so on. The number of cores licensed for a product must be greater than or equal to the number of cores available on that server machine, whether the server is a physical or virtual machine. For example, if a server has eight cores (an octa-core processor), you must purchase at least one 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 one 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, would have less processing speed than if all available cores on the server were utilized.

Note: Each Altova server product license can be used for only one client machine at a time, even if the license has unused licensing capacity. (A client machine is the machine on which the Altova server product is installed.) For example, if a 10-core license is used for a client machine that has 6 CPU cores, then the remaining 4 cores of licensing capacity cannot be used simultaneously for another client machine.

Single-thread execution

If an Altova server product allows single-thread execution, an option for *Single-thread execution* will be available. In these cases, if an Altova server-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, because 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 in LicenseServer, select the *Limit to single thread execution* check box for that product.

Estimate of core requirements
There are various external factors that influence the data volumes and processing times your server can handle (for example: the hardware, the current load on the CPU, and memory allocation of other applications running on the server). In order to measure performance as accurately as possible, test the applications in your environment with data volumes and in conditions that approximate as closely as possible to real business situations.

© 2017-2023 Altova GmbH

3.3 Setup on macOS

This section describes the installation[39] and licensing[41] of RaptorXML Server on macOS systems.

System Requirements (macOS)

- macOS 11 or newer

Prerequisites

- Ensure that Altova LicenseServer has been installed and is running
- Perform installation either as **root** user or as a user with **sudo** privileges.
- The previous version of RaptorXML Server must be uninstalled before a new one is installed.
- If you plan to use Altova's Charts functionality, then at least one font must be installed on your system to ensure that charts will be rendered correctly. To list installed fonts, use, for example, the **fc-list** command of the Fontconfig library.
- The macOS machine must be configured so that its name resolves to an IP address. This means that you must be able to successfully ping the host name from the Terminal using the command **ping <hostname>**.

3.3.1 Installing on macOS

RaptorXML Server is available for installation on macOS systems. Its installation and setup procedure is described below.

Integration with FlowForce

If you are installing RaptorXML Server together with FlowForce Server, it is recommended that you install FlowForce Server first. If you install RaptorXML Server before FlowForce Server, then, after having installed both RaptorXML Server and FlowForce Server, run the following command:

```
cp /usr/local/Altova/RaptorXMLServer2024/etc/*.tool /usr/local/Altova/FlowForceServer2024/tools
```

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

Uninstall RaptorXML Server

Before uninstalling RaptorXML Server, stop the service with the following command:

```
sudo launchctl unload /Library/LaunchDaemons/com.altova.RaptorXMLServer2024.plist
```

To check whether the service has been stopped, open the Activity Monitor in Finder and make sure that RaptorXML Server is not in the list.In the Applications folder in Finder, right-click the RaptorXML 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 following command:

```
sudo rm -rf /usr/local/Altova/RaptorXMLServer2024/
```

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 in Finder and make sure that LicenseServer is not in the list. Then proceed to uninstall in the same way as described above for RaptorXML Server.

Install RaptorXML Server

1. Download the disk image (`.dmg`) file of RaptorXML Server from the Altova website (http://www.altova.com/download.html).
2. Click to open the downloaded disk image (`.dmg`). This causes the RaptorXML Server installer to appear as a new virtual drive on your computer.
3. On the new virtual drive, double-click the installer package (`.pkg`).
4. 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. *See also* *Licensing RaptorXML Server* 41.
5. To eject the drive after installation, right-click it and select **Eject**.

The RaptorXML Server package will be installed in the folder:

`/usr/local/Altova/RaptorXMLServer2024` (application binaries)
`/var/Altova/RaptorXMLServer` (data files: database and logs)

The RaptorXML Server server daemon starts automatically after installation and a re-boot of the machine. You can always start RaptorXML Server as a daemon with the following command:

```
sudo launchctl load /Library/LaunchDaemons/com.altova.RaptorXMLServer2024.plist
```

3.3.2 Installing LicenseServer (macOS)

In order for RaptorXML Server to work, it must be licensed via an Altova LicenseServer on your network. The LicenseServer installation package is available on the virtual drive you have mounted in the previous step. To install LicenseServer, double-click the installer package included on the virtual drive and follow the on-screen instructions. You will need to accept the license agreement for installation to proceed.

Altova LicenseServer can also be downloaded and installed separately from the Altova website (http://www.altova.com/download.html).

The LicenseServer package will be installed in the folder:

`/usr/local/Altova/LicenseServer`

For information about how to register RaptorXML Server with Altova LicenseServer and license it, see Licensing on macOS 41.

 © 2017-2023 Altova GmbH

LicenseServer versions

- Altova server products must be licensed either with the version of LicenseServer that is appropriate to the installed RaptorXML Server version or with a later version of LicenseServer.
- The LicenseServer version that is appropriate for a particular version of RaptorXML Server is displayed during the installation of RaptorXML Server. You can install this version of LicenseServer along with RaptorXML Server. Alternatively, 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 Server.
- If you install a new version of RaptorXML Server and if your installed LicenseServer version is older than the appropriate LicenseServer, install the latest version of LicenseServer 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: 3.12

3.3.3 Licensing RaptorXML Server (macOS)

In order to use RaptorXML Server, it must be licensed with Altova LicenseServer. Licensing is a two-step process:

1. **Register RaptorXML Server** with LicenseServer. Registration is done from RaptorXML Server.
2. **Assign a license** to RaptorXML Server from LicenseServer. Download the latest version of LicenseServer from the Altova website, and install it on your local machine or a machine on your network.

These steps are described in this section. For detailed information, see the LicenseServer user manual at the Altova website.

3.3.3.1 Start LicenseServer, RaptorXML Server

This topic describes how to start Altova LicenseServer (LicenseServer for short) and RaptorXML Server. You must have administrator (root) privileges to be able to start these programs, so you should run these commands 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 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 Server

RaptorXML Server server daemon starts automatically after installation and a re-boot of the machine. You can start RaptorXML Server as a daemon with the following command:

```
sudo launchctl load /Library/LaunchDaemons/com.altova.RaptorXMLServer2024.plist
```

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

```
sudo launchctl unload /Library/LaunchDaemons/com.altova.RaptorXMLServer2024.plist
```

3.3.3.2 Register RaptorXML Server

To register RaptorXML Server from the command line interface, use the `licenseserver` command:

```
sudo /usr/local/Altova/RaptorXMLServer2024/bin/RaptorXML 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/RaptorXMLServer2024/bin/RaptorXML 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 Server executable is:

```
/usr/local/Altova/RaptorXMLServer2024/bin/
```

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

Note: For more information about registering Altova products with LicenseServer, see the LicenseServer user manual.

3.3.3.3 License RaptorXML Server

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

The licensing of Altova server products is based on the number of processor cores available on the product machine. For example, a dual-core processor has two cores, a quad-core processor four cores, a hexa-core processor six cores, and so on. The number of cores licensed for a product must be greater than or equal to the number of cores available on that server machine, whether the server is a physical or virtual machine. For example, if a server has eight cores (an octa-core processor), you must purchase at least one 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 one 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, would have less processing speed than if all available cores on the

 © 2017-2023 Altova GmbH

server were utilized.

Note: Each Altova server product license can be used for only one client machine at a time, even if the license has unused licensing capacity. (A client machine is the machine on which the Altova server product is installed.) For example, if a 10-core license is used for a client machine that has 6 CPU cores, then the remaining 4 cores of licensing capacity cannot be used simultaneously for another client machine.

Single-thread execution
If an Altova server product allows single-thread execution, an option for *Single-thread execution* will be available. In these cases, if an Altova server-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, because 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 in LicenseServer, select the *Limit to single thread execution* check box for that product.

Estimate of core requirements
There are various external factors that influence the data volumes and processing times your server can handle (for example: the hardware, the current load on the CPU, and memory allocation of other applications running on the server). In order to measure performance as accurately as possible, test the applications in your environment with data volumes and in conditions that approximate as closely as possible to real business situations.

3.4 Upgrading RaptorXML Server

When you upgrade to a newer version of RaptorXML Server, the license of your previous version will be used automatically for the newer version if, during installation:

- the new version is registered with the same LicenseServer as that with which the previous version of RaptorXML Server was registered
- you accept the license agreement of RaptorXML Server.

The simplest way to carry over a license from the previous version of RaptorXML Server to the newer version is to let the installation process implement the required steps. The relevant steps during the installation process are listed below in the order in which they occur:

1. Let the installer register the new version of RaptorXML Server with the LicenseServer that holds the license used by the older version of RaptorXML Server.
2. Accept the license agreement of RaptorXML Server. (If you do not accept the agreement, the new version will not be installed.)

Note: If you do not register RaptorXML Server with the correct LicenseServer during the installation process, you will need to register and license RaptorXML Server manually with your alternative LicenseServer.

© 2017-2023 Altova GmbH

3.5 Migrating RaptorXML Server to a New Machine

If you want to migrate RaptorXML Server from one machine to another (including across supported platforms), follow the guidelines below.

Migrating RaptorXML Server to a new machine consists of re-assigning the license from the old machine to the new machine. Do this as follows:

1. Install RaptorXML Server on the new machine. If it has already been installed as part of FlowForce Server installation, ignore this step.
2. On the new machine, register RaptorXML Server with Altova LicenseServer.
3. On the old machine, make sure no clients are using the server.
4. Open the Altova LicenseServer administration page. Deactivate the license from the old RaptorXML Server machine and re-assign it to the new machine.

Note: Migrate the server configuration file in order to keep your previous configuration settings.

Note: If you were using XML catalogs on the old machine, migrate these to the new machine.

4 General Procedures

RaptorXML has special options that support XML Catalogs 47 and Altova global resources 53, both of which enhance portability and modularity. You can leverage the use of these features in your environment to considerable advantage.

This section describes the following:

- How to use XML Catalogs 47.
- How to work with Altova global resources 53.
- Security issues 55 related to RaptorXML procedures and how to deal with them.

© 2017-2023 Altova GmbH

4.1 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 47 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 sections Catalog Structure in RaptorXML Server 48 and Customizing your Catalogs 50. The section Variables for System Locations 52 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 47
- Catalog Structure in RaptorXML Server 48
- Customizing your Catalogs 50
- Variables for Windows System Locations 52

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

Installing schemas via Schema Manager

Schema Manager 363 enables you to quickly and conveniently install important schemas and set up catalog files to correctly access these installed schemas. See the Schema Manager 363 section for more information.

If a document is validated against a schema that is not installed but is available via Schema Manager 363, then the installation via Schema Manager will be triggered automatically. However, if the schema package to be installed via Schema Manager contains namespace mappings, then there will be no automatic installation; in this case, you must start Schema Manager, select the package/s you want to install, and run the installation. If, after installation, RaptorXML Server is not able to correctly locate a schema component, then restart RaptorXML Server and try again.

4.1.1 How Catalogs Work

Catalogs can be used to redirect both DTDs and XML Schemas. While the concept behind the mechanisms of both cases is the same, the details are different and are explained below.

DTDs

Catalogs are commonly used to redirect a call to a DTD to a local URI. This is achieved by mapping, in the catalog file, public or system identifiers to the required local URI. So when the `DOCTYPE` declaration in an XML file is read, its public or system identifier locates the required local resource via the catalog file mapping.

For popular schemas, the `PUBLIC` identifier is usually pre-defined, thus requiring only that the URI in the catalog file map the `PUBLIC` identifier to the correct local copy. When the XML document is parsed, the `PUBLIC` identifier in it is read. If this identifier is found in a catalog file, then the corresponding URL in the catalog file will be looked up and the schema will be read from this location. So, for example, if the following SVG file is opened in RaptorXML Server:

```
<?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 width="20" height="20" xml:space="preserve">
  <g style="fill:red; stroke:#000000">
     <rect x="0" y="0" width="15" height="15"/>
     <rect x="5" y="5" width="15" height="15"/>
  </g>
</svg>
```

The catalog is searched for the **PUBLIC** identifier of this SVG file. Let's say the catalog file contains the following entry:

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

In this case, there is a match for the **PUBLIC** identifier. As a result, the lookup for the SVG DTD is redirected to the URL **schemas/svg/svg11.dtd** (which is relative to the catalog file). This is a local file that will be used as the DTD for the SVG file. If there is no mapping for the **Public** ID in the catalog, then the URL in the XML document will be used (in the SVG fie example above, this is the Internet URL: **http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd**).

XML Schemas

In RaptorXML Server, you can also use catalogs with **XML Schemas**. In the XML instance file, the reference to the schema will occur in the **xsi:schemaLocation** attribute of the XML document's top-level element. For example,

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

The value of the **xsi:schemaLocation** attribute has two parts: a namespace part (green above) and a URI part (highlighted). The namespace part is used in the catalog to map to the alternative resource. For example, the following catalog entry redirects the schema reference above to a schema at an alternative location.

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

Normally, the URI part of the **xsi:schemaLocation** attribute's value 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 must 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 attribute's value to be valid.

4.1.2 Catalog Structure in RaptorXML Server

When RaptorXML Server starts, it loads a file called **RootCatalog.xml** (*structure shown in listing below*), which contains a list of catalog files that will be looked up. You can modify this file and enter as many catalog files to

 © 2017-2023 Altova GmbH

look up as you like, each of which is referenced in a **nextCatalog** element. These catalog files are looked up and the URIs in them are resolved according to their mappings.

Listing of RootCatalog.xml

```
<?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"/>
  <!-- Include all catalogs under common schemas folder on the first directory level -->
  <nextCatalog spy:recurseFrom="%CommonSchemasFolder%" catalog="catalog.xml"
spy:depth="1"/>
  <nextCatalog spy:recurseFrom="%ApplicationWritableDataFolder%/pkgs/.cache"
catalog="remapping.xml" spy:depth="0"/>
  <nextCatalog catalog="CoreCatalog.xml"/>
</catalog>
```

The listing above references a custom catalog (named **CustomCatalog.xml**) and a set of catalogs that locate commonly used schemas (such as W3C XML Schemas and the SVG schema).

- **CustomCatalog.xml** is located in the RaptorXML Server application folder's **etc** subfolder. You must create it from a template file named **CustomCatalog_template.xml**. It is a skeleton file in which you can create your own mappings. You can add mappings to **CustomCatalog.xml** for any schema you require that is not addressed by the catalog files in the Common Schemas Folder. Do this by using the supported elements of the OASIS catalog mechanism (*see next section*).
- The Common Schemas Folder (located via the variable **%CommonSchemasFolder%**) contains a set of commonly used schemas. Inside each of these schema folders is a **catalog.xml** file that maps public and/or system identifiers to URIs that point to locally saved copies of the respective schemas.
- **CoreCatalog.xml** is located in the RaptorXML Server application folder, and is used to locate schemas and stylesheets used by RaptorXML Server-specific processes, such as StyleVision Power Stylesheets which are stylesheets used to generate Altova's Authentic View of XML documents.

Note the following:

- During a new installation of the same major version (same or different minor versions), the template file will be replaced by a new template file, but **CustomCatalog.xml** will be left untouched.
- However, if you are installing a new major version over a previous major version, then the previous major version folder will be deleted—together with its **CustomCatalog.xml**. So, if you want to continue using **CustomCatalog.xml**, make sure that you save **CustomCatalog.xml** from the previous major version folder to a safe place. After the new major version has been installed, you can copy the **CustomCatalog.xml** that you saved to the **etc** folder of the new major version and edit it there as required.

Location variables

The variables that are used in **RootCatalog.xml** (*listing above*) have the following values:

%PersonalFolder%	Personal folder of the current user, for example C:\Users\<name>\Documents
%CommonSchemasFolder%	C:\ProgramData\Altova\Common2024\Schemas

% ApplicationWritableDataFolde r%	C:\ProgramData\Altova

Location of catalog files and schemas
Note the locations of the various catalog files.

- **RootCatalog.xml**, **CustomCatalog.xml**, **CustomCatalog_template.xml**, and **CoreCatalog.xml** are in the RaptorXML Server application folder.
- The **catalog.xml** files are each in a specific schema folder, these schema folders being inside the Common Schemas Folder.

4.1.3 Customizing your Catalogs

When creating entries in **CustomCatalog.xml** (or any other catalog file that is to be read by RaptorXML Server), use only the following elements of the OASIS catalog specification. Each of the elements below is listed with an explanation of their attribute values. For a more detailed explanation, see the XML Catalogs specification. Note that each element can take the **xml:base** attribute, which is used to specify the base URI of that element.

- `<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"/>`

Note the following points:

- In cases where there is no public identifier, as with most stylesheets, the system identifier can be directly mapped to a URL via the **system** element.
- 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. For more information on these elements, see the XML Catalogs specification.

From release 2014 onwards, RaptorXML Server adheres closely to the XML Catalogs specification (OASIS Standard V1.1, 7 October 2005) specification. This specification strictly separates external-identifier look-ups (those with a Public ID or System ID) from URI look-ups (URIs that are not Public IDs or System IDs). Namespace URIs must therefore be considered simply URIs—not Public IDs or System IDs—and must be used as URI look-ups rather than external-identifier look-ups. In RaptorXML Server versions prior to version 2014, schema namespace URIs were translated through **<public>** mappings. From version 2014 onwards, **<uri>** mappings have to be used.

Prior to v2014: `<public publicID="http://www.MyMapping.com/ref" uri="file:///C:/MyDocs/Catalog/test.xsd"/>`
V-2014 onwards: `<uri name="http://www.MyMapping.com/ref" uri="file:///C:/MyDocs/Catalog/test.xsd"/>`

 © 2017-2023 Altova GmbH

How RaptorXML Server finds a referenced schema

A schema is referenced in an XML document via the **xsi:scemaLocation** attribute (*shown below*). The value of the **xsi:schemaLocation** attribute has two parts: a namespace part (green) and a URI part (highlighted).

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

The set of steps that is followed to find a referenced schema depends on the validation options **--schemalocation-hints** and **--schema-mapping**. Given below are the procedures for each value of the two options:

- **--schemalocation-hints=load-by-schemalocation | load-by-namespace | load-combining-both | ignore**
  Specifies the behavior of the **xsi:schemaLocation** and **xsi:noNamespaceSchemaLocation** attributes: whether to load a schema document and, if yes, which information should be used to find it; (the default is **load-by-schemalocation**).
  - **load-by-schemalocation**
    1. If the URI part of the **xsi:schemaLocation** is mapped in a catalog, load the resulting URI
    2. Load the URI directly
  - **load-by-namespace**
    1. If the namespace part of the **xsi:schemaLocation** is mapped in a catalog, load the resulting URI.
    2. Load nothing.
  - **load-combining-both**
    1. If the URI part of the **xsi:schemaLocation** is mapped in a catalog, load the resulting URI.
    2. If the namespace part of the **xsi:schemaLocation** is mapped in a catalog, load the resulting URI.
    3. Load the URI part directly.

- **--schema-mapping=prefer-schemalocation | prefer-namespace**
  If schema location and namespace are both used to find a schema document, then this option specifies which of the two should be preferred during catalog lookup; (the default is **prefer-schemalocation**). This option is used to change the order of the first two steps in the **load-combining-both** variant above.

XML Schema specifications

XML Schema specification information is built into RaptorXML Server and the validity of XML Schema (`.xsd`) documents is checked against this internal information. In an XML Schema document, therefore, no references should be made to any schema that defines the XML Schema specification.

The **catalog.xml** file in the `%AltovaCommonSchemasFolder%\Schemas\schema` folder contains references to DTDs that implement older XML Schema specifications. You should not validate your XML Schema documents against these schemas. The referenced files are included solely to provide RaptorXML Server with entry helper info for editing purposes should you wish to create documents according to these older recommendations.

4.1.4 Variables for Windows System Locations

Shell environment variables can be used in the `nextCatalog` element to specify the path to various system locations (*see RootCatalog.xml listing above*). The following shell environment variables are supported:

`%PersonalFolder%`	Full path to the Personal folder of the current user, for example `C:\Users\<name>\Documents`
`%CommonSchemasFolder%`	`C:\ProgramData\Altova\Common2024\Schemas`
`%ApplicationWritableDataFolder%`	`C:\ProgramData\Altova`
`%AltovaCommonFolder%`	C:\Program Files\Altova\Common2024
`%DesktopFolder%`	Full path to the Desktop folder of the current user.
`%ProgramMenuFolder%`	Full path to the Program Menu folder of the current user.
`%StartMenuFolder%`	Full path to Start Menu folder of the current user.
`%StartUpFolder%`	Full path to Start Up folder of the current user.
`%TemplateFolder%`	Full path to the Template folder of the current user.
`%AdminToolsFolder%`	Full path to the file system directory that stores administrative tools of the current user.
`%AppDataFolder%`	Full path to the Application Data folder of the current user.
`%CommonAppDataFolder%`	Full path to the file directory containing application data of all users.
`%FavoritesFolder%`	Full path of the Favorites folder of the current user.
`%PersonalFolder%`	Full path to the Personal folder of the current user.
`%SendToFolder%`	Full path to the SendTo folder of the current user.
`%FontsFolder%`	Full path to the System Fonts folder.
`%ProgramFilesFolder%`	Full path to the Program Files folder of the current user.
`%CommonFilesFolder%`	Full path to the Common Files folder of the current user.
`%WindowsFolder%`	Full path to the Windows folder of the current user.
`%SystemFolder%`	Full path to the System folder of the current user.
`%LocalAppDataFolder%`	Full path to the file system directory that serves as the data repository for local (nonroaming) applications.
`%MyPicturesFolder%`	Full path to the MyPictures folder.

© 2017-2023 Altova GmbH

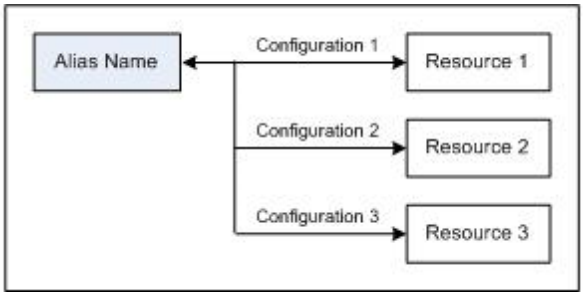
4.2 Global Resources

This section:

- About global resources 53
- Using global resources 53

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 the resource can be changed merely by switching the name of the 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:

```
raptorxml 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:

```
raptorxml 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:

```
raptorxml 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.

© 2017-2023 Altova GmbH

4.3 Security Issues

This section:

- Security concerns related to the HTTP interface 55
- Making Python scripts safe 55

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

Security concerns related to the HTTP REST interface

The HTTP REST 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 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` 243 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 Server, the script will only work if it is located in the trusted directory 243. 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` 243 setting of the server configuration file 242, 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 243 (which is a sub-directory of the `output-root-directory` 243), 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 243 for potential vulnerability issues.

5 Command Line Interface (CLI)

The RaptorXML Server executable provides application functionality that can be called from the command line interface (CLI). The path to the executable is:

Linux	`/opt/Altova/RaptorXMLServer2024/bin/`**`raptorxml`**
Mac	`/usr/local/Altova/RaptorXMLServer2024/bin/`**`raptorxml`**
Windows	`<ProgramFilesFolder>\Altova\RaptorXMLServer2024\bin\`**`RaptorXML.exe`**

Usage

The command line syntax is:

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

- `--help` (short form `--h`) displays the help text of the given command. If no command is named, then all commands of the executable are listed, each with a brief description of the command.
- `--version` displays the version number of RaptorXML Server.
- `<command>` is the command to execute. Commands are described in the sub-sections of this section *(see list below)*.
- `[options]` are the options of a command; they are listed and described with their respective commands.
- `[arguments]` are the arguments of a command; they are listed and described with their respective commands.

▼ Casing and slashes on the command line

`RaptorXML` (and **`RaptorXMLServer`** for administration commands) *on Windows*
`raptorxml` (and **`raptorxmlserver`** for administration commands) *on Windows and Unix (Linux, Mac)*

* Note that lowercase (`raptorxml` and `raptorxmlserver`) works on all platforms (Windows, Linux, and Mac), while upper-lower (`RaptorXML`) works only on Windows and Mac.
* Use forward slashes on Linux and Mac, backslashes on Windows.

CLI commands

The commands have been organized by their functionality as listed below, and are described in the sub-sections of this section.

- XML, DTD, XSD Validation Commands 58
- Well-formedness Check Commands 80
- XQuery Commands 92
- XSLT Commands 121
- JSON/Avro Commands 136
- XML Signature Commands 183
- General Commands 195

 © 2017-2023 Altova GmbH

- Localization Commands 199
- License Commands 202
- Administration Commands 207

5.1 XML, DTD, XSD Validation Commands

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

- valxml-withdtd 58: Validates an XML instance document against a DTD
- valxml-withxsd 62: Validates an XML instance document against an XML Schema
- valdtd 69: Validates a DTD document
- valxsd 73: Validates a W3C XML Schema (XSD) document.

5.1.1 valxml-withdtd (xml)

The **valxml-withdtd | xml** command validates one or more XML instance documents against a DTD.

```
raptorxml 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` 223 option set to `true` (*see the Options list below*).

Examples

Examples of the **valxml-withdtd** command:

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

▼ Casing and slashes on the command line

RaptorXML (and **RaptorXMLServer** for administration commands) *on Windows*
raptorxml (and **raptorxmlserver** for administration commands) *on Windows and Unix (Linux, Mac)*

* Note that lowercase (`raptorxml` and `raptorxmlserver`) works on all platforms (Windows, Linux, and Mac), while upper-lower (`RaptorXML`) works only on Windows and Mac.
* Use forward slashes on Linux and Mac, backslashes on Windows.

▼ Backslashes, spaces, and special characters on Windows systems

On Windows systems: When spaces or special characters occur in strings (for example in file or folder names, or company, person or product names), use quotes: for example, **"My File"**. Note, however, that a backslash followed by a double-quotation mark (for example, **"C:\My directory\"**) might not be read correctly. This is because the backslash character is also used to indicate the start of an escape sequence, and the escape sequence `\"` stands for the double-quotation mark character. If you want to escape this sequence of characters, use a preceding backslash, like this: `\\"`. To summarize: If you

 © 2017-2023 Altova GmbH

need to write a file path that contains spaces or an end backslash, write it like this: `"C:\My Directory\ \"`.

Options

Options are listed in short form (if available) and long form. You can use one or two dashes for both short and long forms. An option may or may not take a value. If it takes a value, it is written like this: **`--option=value`**. 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. If an option takes a Boolean value and no value is specified, then the option's default value is `TRUE`. Use the **`--h, --help`** option to display information about the command.

- 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 sub-directories, including in ZIP archives. 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. References to ZIP files must be given in quotes. The wildcard characters `*` and `?` may be used. So, `*.xml` will select all `.xml` files in the (zip) folder. The option'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\RaptorXMLServer2024\etc\RootCatalog.xml`). See the section, XML Catalogs [47], 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 [47], for information about working with catalogs.

  - enable-globalresources

    `--enable-globalresources =` `true|false`
    Enables global resources [53]. 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 [53] (and enables global resources) [53].

  - globalresourcefile [gr]

    `--gr | --globalresourcefile =` *FILE*
    Specifies the global resource file [53] (and enables global resources) [53].

- Common options
  - 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 | unlimited*
    Specifies the error limit with a value range of `1` to `9999` or `unlimited`. The default value is `100`. Processing stops when the error limit is reached. Useful for limiting processor use during validation/transformation.

© 2017-2023 Altova GmbH

- info-limit

  --info-limit = *N | unlimited*
  Specifies the information message limit in the range `1-65535` or `unlimited`. Processing continues if the specified info limit is reached, but further messages are not reported. The default value is `100`.

- 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`.)

- 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.

- log-output

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

- network-timeout

  --network-timeout = *VALUE*
  Specifies the timeout in milliseconds for remote I/O operations. Default is: `40000`.

- recurse

  --recurse = `true|false`
  Used to select files within sub-directories, including in ZIP archives. 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. References to ZIP files must be given in quotes. The wildcard characters `*` and `?` may be used. So, `*.xml` will select all `.xml` files in the (zip) folder. The option's default value is `false`.
  Note: Boolean option values are set to `true` if the option is specified without a value.

- 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 Server. If used with a command, place `--version` before the command.

▼ warning-limit

--warning-limit = *N | unlimited*
Specifies the warning limit in the range `1-65535` or `unlimited`. Processing continues if this limit is reached, but further warnings are not reported. The default value is `100`.

5.1.2 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.

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

- The *InputFile* argument is the XML document to validate. The `--schemalocation-hints` [225] option specifies what mechanism is used to find the schema. The `--xsd=FILE` [224] option specifies the schema/s to use if the XML file contains no schema reference.
- 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` [223] option set to `true` (*see the Options list below*).

Note: If using the `--script` option to run Python scripts [352], make sure to also specify `--streaming=false`.

Examples

Examples of the **valxml-withxsd** command:

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

▼ Casing and slashes on the command line

RaptorXML (and **RaptorXMLServer** for administration commands) *on Windows*
raptorxml (and **raptorxmlserver** for administration commands) *on Windows and Unix (Linux, Mac)*

* Note that lowercase (`raptorxml` and `raptorxmlserver`) works on all platforms (Windows, Linux, and Mac), while upper-lower (`RaptorXML`) works only on Windows and Mac.
* Use forward slashes on Linux and Mac, backslashes on Windows.

▼ Backslashes, spaces, and special characters on Windows systems

 © 2017-2023 Altova GmbH

On Windows systems: When spaces or special characters occur in strings (for example in file or folder names, or company, person or product names), use quotes: for example, **"My File"**. Note, however, that a backslash followed by a double-quotation mark (for example, **"C:\My directory\"**) might not be read correctly. This is because the backslash character is also used to indicate the start of an escape sequence, and the escape sequence **\"** stands for the double-quotation mark character. If you want to escape this sequence of characters, use a preceding backslash, like this: **\\"**. To summarize: If you need to write a file path that contains spaces or an end backslash, write it like this: **"C:\My Directory\ \"**.

Options

Options are listed in short form (if available) and long form. You can use one or two dashes for both short and long forms. An option may or may not take a value. If it takes a value, it is written like this: **--option=value**. 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. If an option takes a Boolean value and no value is specified, then the option's default value is `TRUE`. Use the **--h, --help** option to display information about the command.

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.

ct-restrict-mode

--ct-restrict-mode = `1.0|1.1|default`
Specifies how to check complex type restrictions. A value of **1.0** checks complex type restrictions as defined in the XSD 1.0 specification—even in XSD 1.1 validation mode. A value of **1.1** checks complex type restrictions as defined in the XSD 1.1 specification—even in XSD 1.0 validation mode. A value of **default** checks complex type restrictions as defined in the XSD specification of the current validation mode (1.0 or 1.1). The default value is **default**.

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 sub-directories, including in ZIP archives. 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. References to ZIP files must be given in quotes. The wildcard characters `*` and `?` may be used. So, `*.xml` will select all `.xml` files in the (zip) folder. The option's default value is `false`.
  Note: Boolean option values are set to `true` if the option is specified without a value.

- report-import-namespace-mismatch-as-warning

  `--report-import-namespace-mismatch-as-warning =` `true|false`
  Downgrades namespace or target-namespace mismatch errors when importing schemas with **`xs:import`** from errors to warnings. The 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 option specifies whether to load a schema document or just license a namespace, and, if a schema document is to be loaded, which information should be used to find it. Default: `load-preferring-schemalocation`.
  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 [47]. 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 [47]. If no `schemaLocation` attribute is present, then the value of the `namespace` attribute is used via a catalog mapping [47]. This is the **default value**.
  - `load-by-namespace`: The value of the `namespace` attribute is used to locate the schema via a catalog mapping [47].
  - `load-combining-both`: If either the `namespace` or `schemaLocation` attribute has a catalog mapping [47], then the mapping is used. If both have catalog mappings [47], then the value of the `--schema-mapping` option (XML/XSD option [225]) decides which mapping is used. If no catalog mapping [47] 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`
  Specifies the behavior of the `xsi:schemaLocation` and `xsi:noNamespaceSchemaLocation`

 © 2017-2023 Altova GmbH

attributes: Whether to load a schema document, and, if yes, which information should be used to find it. Default: `load-by-schemalocation`.

- The `load-by-schemalocation` value uses the URL of the schema location[383] in the `xsi:schemaLocation` and `xsi:noNamespaceSchemaLocation` attributes in XML instance documents. This is the **default value**.
- The `load-by-namespace` value takes the namespace part[383] of `xsi:schemaLocation` and an empty string in the case of `xsi:noNamespaceSchemaLocation` and locates the schema via a catalog mapping[47].
- If `load-combining-both` is used and if either the namespace part or the URL part has a catalog mapping[47], then the catalog mapping[47] is used. If both have catalog mappings[47], then the value of the --schema-mapping option (XML/XSD option[225]) 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 schema location and namespace are both used to find a schema document, specifies which of them should be preferred during catalog lookup. (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[47], 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. Add the option multiple times to specify more than one script.

▼ script-api-version

`--api, --script-api-version =` `1; 2; 2.1 to 2.4; 2.4.1; 2.5 to 2.8; 2.8.1 to 2.8.6`

Specifies the Python API version to be used for the script. The default value is the latest version, currently **`2.8.6`**. Instead of integer values such as `1` and `2`, you can also use the corresponding values `1.0` and `2.0`. Similarly, you can use the three-digit `2.5.0` for the two-digit `2.5`. Also see the topic Python API Versions[353].

▼ script-output

`--script-output =` *`FILE`*

Writes the script's standard output to the file named in *`FILE`*.

▼ script-param

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

Additional user-specified parameters that can be accessed during the execution of Python scripts. Add the option multiple times to specify more than one script parameter.

▼ 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 for the XML instance document: wf=wellformed check; id=wellformed with ID/IDREF checks; valid=validation. Default value is wf. Note that a value of valid requires that each instance document loaded during processing references a DTD. If no DTD exists, an error is reported.

▼ xml-mode-for-schemas

--xml-mode-for-schemas = wf|id|valid
Specifies the XML processing mode to use for XML schema documents: wf=wellformed check; id=wellformed with ID/IDREF checks; valid=validation. Default value is wf. Note that a value of valid requires that each schema document loaded during processing references a DTD. If no DTD exists, an error is reported.

▼ 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 more than one schema document.

▼ 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

 © 2017-2023 Altova GmbH

--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\RaptorXMLServer2024\etc\RootCatalog.xml`). See the section, XML Catalogs [47], 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 [47], for information about working with catalogs.

- enable-globalresources

  --enable-globalresources = `true|false`
  Enables global resources [53]. 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 [53] (and enables global resources) [53].

- globalresourcefile [gr]

  --gr | --globalresourcefile = *FILE*
  Specifies the global resource file [53] (and enables global resources) [53].

- Common options
  - 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 | unlimited*
    Specifies the error limit with a value range of `1` to `9999` or `unlimited`. The default value is `100`. Processing stops when the error limit is reached. Useful for limiting processor use during validation/transformation.

  - info-limit

    --info-limit = *N | unlimited*
    Specifies the information message limit in the range `1-65535` or `unlimited`. Processing continues if the specified info limit is reached, but further messages are not reported. The default value is `100`.

  - 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`.)

- 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.

- log-output

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

- network-timeout

  `--network-timeout =` *`VALUE`*
  Specifies the timeout in milliseconds for remote I/O operations. Default is: `40000`.

- recurse

  `--recurse =` `true|false`
  Used to select files within sub-directories, including in ZIP archives. 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. References to ZIP files must be given in quotes. The wildcard characters `*` and `?` may be used. So, `*.xml` will select all `.xml` files in the (zip) folder. The option's default value is `false`.
  Note: Boolean option values are set to `true` if the option is specified without a value.

- 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 Server. If used with a command, place `--version` before the command.

- warning-limit

  `--warning-limit =` *`N | unlimited`*
  Specifies the warning limit in the range `1-65535` or `unlimited`. Processing continues if this limit is reached, but further warnings are not reported. The default value is `100`.

 © 2017-2023 Altova GmbH

5.1.3 valdtd (dtd)

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

```
raptorxml 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` [223] option set to `true` (*see the Options list below*).

Examples

Examples of the **valdtd** command:

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

▼ Casing and slashes on the command line

RaptorXML (and **RaptorXMLServer** for administration commands) *on Windows*
raptorxml (and **raptorxmlserver** for administration commands) *on Windows and Unix (Linux, Mac)*

* Note that lowercase (`raptorxml` and `raptorxmlserver`) works on all platforms (Windows, Linux, and Mac), while upper-lower (`RaptorXML`) works only on Windows and Mac.
* Use forward slashes on Linux and Mac, backslashes on Windows.

▼ Backslashes, spaces, and special characters on Windows systems

On Windows systems: When spaces or special characters occur in strings (for example in file or folder names, or company, person or product names), use quotes: for example, **"My File"**. Note, however, that a backslash followed by a double-quotation mark (for example, **"C:\My directory\"**) might not be read correctly. This is because the backslash character is also used to indicate the start of an escape sequence, and the escape sequence **\"** stands for the double-quotation mark character. If you want to escape this sequence of characters, use a preceding backslash, like this: **\\"**. To summarize: If you need to write a file path that contains spaces or an end backslash, write it like this: **"C:\My Directory\ \"**.

Options

Options are listed in short form (if available) and long form. You can use one or two dashes for both short and

long forms. An option may or may not take a value. If it takes a value, it is written like this: **`--option=value`**. 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. If an option takes a Boolean value and no value is specified, then the option's default value is `TRUE`. Use the **`--h, --help`** option to display information about the command.

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 sub-directories, including in ZIP archives. 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. References to ZIP files must be given in quotes. The wildcard characters `*` and `?` may be used. So, `*.xml` will select all `.xml` files in the (zip) folder. The option's default value is `false`.
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. Add the option multiple times to specify more than one script.

script-api-version

`--api, --script-api-version =` `1; 2; 2.1 to 2.4; 2.4.1; 2.5 to 2.8; 2.8.1 to 2.8.6`
Specifies the Python API version to be used for the script. The default value is the latest version, currently **`2.8.6`**. Instead of integer values such as `1` and `2`, you can also use the corresponding values `1.0` and `2.0`. Similarly, you can use the three-digit `2.5.0` for the two-digit `2.5`. Also see the topic Python API Versions 353.

script-output

`--script-output =` *`FILE`*
Writes the script's standard output to the file named in *`FILE`*.

script-param

`--script-param =` *`KEY:VALUE`*
Additional user-specified parameters that can be accessed during the execution of Python scripts. Add the option multiple times to specify more than one script parameter.

Catalogs and global resources

 © 2017-2023 Altova GmbH

- 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\RaptorXMLServer2024\etc\RootCatalog.xml`). See the section, XML Catalogs [47], 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 [47], for information about working with catalogs.

- enable-globalresources

  --enable-globalresources = `true|false`
  Enables global resources [53]. 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 [53] (and enables global resources) [53].

- globalresourcefile [gr]

  --gr | --globalresourcefile = *FILE*
  Specifies the global resource file [53] (and enables global resources) [53].

- Common options
  - 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 | unlimited*
    Specifies the error limit with a value range of `1` to `9999` or `unlimited`. The default value is `100`. Processing stops when the error limit is reached. Useful for limiting processor use during validation/transformation.

  - info-limit

    --info-limit = *N | unlimited*
    Specifies the information message limit in the range `1-65535` or `unlimited`. Processing continues if the specified info limit is reached, but further messages are not reported. The default value is `100`.

  - 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`.)

- 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.

- log-output

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

- network-timeout

  `--network-timeout =` *`VALUE`*
  Specifies the timeout in milliseconds for remote I/O operations. Default is: `40000`.

- recurse

  `--recurse =` `true|false`
  Used to select files within sub-directories, including in ZIP archives. 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. References to ZIP files must be given in quotes. The wildcard characters `*` and `?` may be used. So, `*.xml` will select all `.xml` files in the (zip) folder. The option's default value is `false`.
  Note: Boolean option values are set to `true` if the option is specified without a value.

- 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 Server. If used with a command, place `--version` before the command.

- warning-limit

  `--warning-limit =` *`N | unlimited`*
  Specifies the warning limit in the range `1-65535` or `unlimited`. Processing continues if this limit is

 © 2017-2023 Altova GmbH

reached, but further warnings are not reported. The default value is `100`.

5.1.4 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.

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

- The *`InputFile`* argument is the XML Schema document to validate. The `--xsd-version=1.0|1.1|detect`[225] 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`[223] option set to `true` (*see the Options list below*).

Examples

Examples of the **`valxsd`** command:

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

▼ Casing and slashes on the command line

`RaptorXML` (and **`RaptorXMLServer`** for administration commands) *on Windows*
`raptorxml` (and **`raptorxmlserver`** for administration commands) *on Windows and Unix (Linux, Mac)*

* Note that lowercase (`raptorxml` and `raptorxmlserver`) works on all platforms (Windows, Linux, and Mac), while upper-lower (`RaptorXML`) works only on Windows and Mac.
* Use forward slashes on Linux and Mac, backslashes on Windows.

▼ Backslashes, spaces, and special characters on Windows systems

On Windows systems: When spaces or special characters occur in strings (for example in file or folder names, or company, person or product names), use quotes: for example, **`"My File"`**. Note, however, that a backslash followed by a double-quotation mark (for example, **`"C:\My directory\"`**) might not be read correctly. This is because the backslash character is also used to indicate the start of an escape sequence, and the escape sequence **`\"`** stands for the double-quotation mark character. If you want to escape this sequence of characters, use a preceding backslash, like this: **`\\"`**. To summarize: If you need to write a file path that contains spaces or an end backslash, write it like this: **`"C:\My Directory\\"`**.

Options

Options are listed in short form (if available) and long form. You can use one or two dashes for both short and long forms. An option may or may not take a value. If it takes a value, it is written like this: **`--option=value`**. 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. If an option takes a Boolean value and no value is specified, then the option's default value is `TRUE`. Use the **`--h, --help`** option to display information about the command.

Validation and processing

ct-restrict-mode

`--ct-restrict-mode =` `1.0|1.1|default`

Specifies how to check complex type restrictions. A value of **`1.0`** checks complex type restrictions as defined in the XSD 1.0 specification—even in XSD 1.1 validation mode. A value of **`1.1`** checks complex type restrictions as defined in the XSD 1.1 specification—even in XSD 1.0 validation mode. A value of **`default`** checks complex type restrictions as defined in the XSD specification of the current validation mode (1.0 or 1.1). The default value is **`default`**.

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 sub-directories, including in ZIP archives. 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. References to ZIP files must be given in quotes. The wildcard characters `*` and `?` may be used. So, `*.xml` will select all `.xml` files in the (zip) folder. The option's default value is `false`.

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

report-import-namespace-mismatch-as-warning

`--report-import-namespace-mismatch-as-warning =` `true|false`

Downgrades namespace or target-namespace mismatch errors when importing schemas with **`xs:import`** from errors to warnings. The 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 option specifies whether to load a schema document or just

 © 2017-2023 Altova GmbH

license a namespace, and, if a schema document is to be loaded, which information should be used to find it. Default: `load-preferring-schemalocation`.
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[47]. 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[47]. If no `schemaLocation` attribute is present, then the value of the `namespace` attribute is used via a catalog mapping[47]. This is the **default value**.
- `load-by-namespace`: The value of the `namespace` attribute is used to locate the schema via a catalog mapping[47].
- `load-combining-both`: If either the `namespace` or `schemaLocation` attribute has a catalog mapping[47], then the mapping is used. If both have catalog mappings[47], then the value of the `--schema-mapping` option (XML/XSD option[225]) decides which mapping is used. If no catalog mapping[47] 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`
Specifies the behavior of the `xsi:schemaLocation` and `xsi:noNamespaceSchemaLocation` attributes: Whether to load a schema document, and, if yes, which information should be used to find it. Default: `load-by-schemalocation`.

- The `load-by-schemalocation` value uses the URL of the schema location[383] in the `xsi:schemaLocation` and `xsi:noNamespaceSchemaLocation` attributes in XML instance documents. This is the **default value**.
- The `load-by-namespace` value takes the namespace part[383] of `xsi:schemaLocation` and an empty string in the case of `xsi:noNamespaceSchemaLocation` and locates the schema via a catalog mapping[47].
- If `load-combining-both` is used and if either the namespace part or the URL part has a catalog mapping[47], then the catalog mapping[47] is used. If both have catalog mappings[47], then the value of the --schema-mapping option (XML/XSD option[225]) 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 schema location and namespace are both used to find a schema document, specifies which of them should be preferred during catalog lookup. (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[47], 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. Add the option multiple times to specify more than one script.

- script-api-version

  `--api, --script-api-version =` `1; 2; 2.1 to 2.4; 2.4.1; 2.5 to 2.8; 2.8.1 to 2.8.6`
  Specifies the Python API version to be used for the script. The default value is the latest version, currently **`2.8.6`**. Instead of integer values such as `1` and `2`, you can also use the corresponding values `1.0` and `2.0`. Similarly, you can use the three-digit `2.5.0` for the two-digit `2.5`. Also see the topic Python API Versions 353.

- script-output

  `--script-output =` *`FILE`*
  Writes the script's standard output to the file named in *`FILE`*.

- script-param

  `--script-param =` *`KEY:VALUE`*
  Additional user-specified parameters that can be accessed during the execution of Python scripts. Add the option multiple times to specify more than one script parameter.

- 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-for-schemas

  `--xml-mode-for-schemas =` `wf|id|valid`
  Specifies the XML processing mode to use for XML schema documents: `wf`=wellformed check; `id`=wellformed with ID/IDREF checks; `valid`=validation. Default value is `wf`. Note that a value of `valid` requires that each schema document loaded during processing references a DTD. If no DTD exists, an error is reported.

- 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`*

 © 2017-2023 Altova GmbH

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\RaptorXMLServer2024\etc\RootCatalog.xml`). See the section, XML Catalogs[47], 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[47], for information about working with catalogs.

- enable-globalresources

  `--enable-globalresources =` `true|false`
  Enables global resources[53]. 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[53] (and enables global resources)[53].

- globalresourcefile [gr]

  `--gr | --globalresourcefile =` *FILE*
  Specifies the global resource file[53] (and enables global resources)[53].

- Common options
  - 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 | unlimited*
    Specifies the error limit with a value range of `1` to `9999` or `unlimited`. The default value is `100`. Processing stops when the error limit is reached. Useful for limiting processor use during validation/transformation.

  - info-limit

    `--info-limit =` *N | unlimited*
    Specifies the information message limit in the range `1-65535` or `unlimited`. Processing continues if the specified info limit is reached, but further messages are not reported. The default value is `100`.

  - 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`.)

- 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.

- log-output

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

- network-timeout

  --network-timeout = *VALUE*
  Specifies the timeout in milliseconds for remote I/O operations. Default is: `40000`.

- recurse

  --recurse = `true|false`
  Used to select files within sub-directories, including in ZIP archives. 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. References to ZIP files must be given in quotes. The wildcard characters `*` and `?` may be used. So, `*.xml` will select all `.xml` files in the (zip) folder. The option's default value is `false`.
  Note: Boolean option values are set to `true` if the option is specified without a value.

- 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 Server. If used with a command, place `--version` before the command.

- warning-limit

  --warning-limit = *N | unlimited*
  Specifies the warning limit in the range `1-65535` or `unlimited`. Processing continues if this limit is reached, but further warnings are not reported. The default value is `100`.

© 2017-2023 Altova GmbH

5.2 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 80: Checks the well-formedness of XML documents
- wfdtd 84: Checks the well-formedness of DTDs
- wfany 87: Checks the well-formedness of an XML document or DTD. Type is detected automatically

5.2.1 wfxml

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

```
raptorxml wfxml [options] InputFile
```

- The *InputFile* argument is the XML document to check for well-formedness.
- 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` 223 option set to `true` (*see the Options list below*).

Examples

Examples of the **wfxml** command:

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

▼ Casing and slashes on the command line

RaptorXML (and **RaptorXMLServer** for administration commands) *on Windows*
raptorxml (and **raptorxmlserver** for administration commands) *on Windows and Unix (Linux, Mac)*

* Note that lowercase (`raptorxml` and `raptorxmlserver`) works on all platforms (Windows, Linux, and Mac), while upper-lower (`RaptorXML`) works only on Windows and Mac.
* Use forward slashes on Linux and Mac, backslashes on Windows.

▼ Backslashes, spaces, and special characters on Windows systems

On Windows systems: When spaces or special characters occur in strings (for example in file or folder names, or company, person or product names), use quotes: for example, **"My File"**. Note, however, that a backslash followed by a double-quotation mark (for example, **"C:\My directory\"**) might not be read correctly. This is because the backslash character is also used to indicate the start of an escape sequence, and the escape sequence **\"** stands for the double-quotation mark character. If you want to escape this sequence of characters, use a preceding backslash, like this: **\\"**. To summarize: If you need to write a file path that contains spaces or an end backslash, write it like this: **"C:\My Directory**

 © 2017-2023 Altova GmbH

\".

Options

Options are listed in short form (if available) and long form. You can use one or two dashes for both short and long forms. An option may or may not take a value. If it takes a value, it is written like this: **--option=value**. 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. If an option takes a Boolean value and no value is specified, then the option's default value is `TRUE`. Use the **--h, --help** option to display information about the command.

- 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 sub-directories, including in ZIP archives. 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. References to ZIP files must be given in quotes. The wildcard characters `*` and `?` may be used. So, `*.xml` will select all `.xml` files in the (zip) folder. The option'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\RaptorXMLServer2024\etc\RootCatalog.xml`). See the section, XML Catalogs [47], 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 [47], for information about working with catalogs.

  - enable-globalresources

    `--enable-globalresources =` `true|false`
    Enables global resources [53]. 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 [53] (and enables global resources) [53].

  - globalresourcefile [gr]

    `--gr | --globalresourcefile =` *FILE*
    Specifies the global resource file [53] (and enables global resources) [53].

- Common options
  - 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 | unlimited*
    Specifies the error limit with a value range of `1` to `9999` or `unlimited`. The default value is `100`. Processing stops when the error limit is reached. Useful for limiting processor use during validation/transformation.

  - info-limit

© 2017-2023 Altova GmbH

--info-limit = *N | unlimited*
Specifies the information message limit in the range `1-65535` or `unlimited`. Processing continues if the specified info limit is reached, but further messages are not reported. The default value is `100`.

- 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`.)

- 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.

- log-output

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

- network-timeout

  --network-timeout = *VALUE*
  Specifies the timeout in milliseconds for remote I/O operations. Default is: `40000`.

- recurse

  --recurse = `true|false`
  Used to select files within sub-directories, including in ZIP archives. 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. References to ZIP files must be given in quotes. The wildcard characters `*` and `?` may be used. So, `*.xml` will select all `.xml` files in the (zip) folder. The option's default value is `false`.
  Note: Boolean option values are set to `true` if the option is specified without a value.

- 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 Server. If used with a command, place `--version` before the command.

▼ warning-limit

`--warning-limit =` *`N | unlimited`*
Specifies the warning limit in the range `1-65535` or `unlimited`. Processing continues if this limit is reached, but further warnings are not reported. The default value is `100`.

5.2.2 wfdtd

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

```
raptorxml wfdtd [options] InputFile
```

- The *`InputFile`* argument is the DTD document to check for well-formedness.
- 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` 223 option set to `true` (*see the Options list below*).

Examples

Examples of the **`wfdtd`** command:

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

▼ Casing and slashes on the command line

`RaptorXML` (and **`RaptorXMLServer`** for administration commands) *on Windows*
`raptorxml` (and **`raptorxmlserver`** for administration commands) *on Windows and Unix (Linux, Mac)*

* Note that lowercase (`raptorxml` and `raptorxmlserver`) works on all platforms (Windows, Linux, and Mac), while upper-lower (`RaptorXML`) works only on Windows and Mac.
* Use forward slashes on Linux and Mac, backslashes on Windows.

▼ Backslashes, spaces, and special characters on Windows systems

On Windows systems: When spaces or special characters occur in strings (for example in file or folder names, or company, person or product names), use quotes: for example, **`"My File"`**. Note, however, that a backslash followed by a double-quotation mark (for example, **`"C:\My directory\"`**) might not be read correctly. This is because the backslash character is also used to indicate the start of an escape sequence, and the escape sequence **`\"`** stands for the double-quotation mark character. If you want to

 © 2017-2023 Altova GmbH

escape this sequence of characters, use a preceding backslash, like this: `\\"`. To summarize: If you need to write a file path that contains spaces or an end backslash, write it like this: `"C:\My Directory\ \"`.

Options

Options are listed in short form (if available) and long form. You can use one or two dashes for both short and long forms. An option may or may not take a value. If it takes a value, it is written like this: **`--option=value`**. 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. If an option takes a Boolean value and no value is specified, then the option's default value is `TRUE`. Use the **`--h, --help`** option to display information about the command.

- 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 sub-directories, including in ZIP archives. 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. References to ZIP files must be given in quotes. The wildcard characters `*` and `?` may be used. So, `*.xml` will select all `.xml` files in the (zip) folder. The option'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\RaptorXMLServer2024\etc\RootCatalog.xml`). See the section, XML Catalogs 47, 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 47, for information about working with catalogs.

  - enable-globalresources

--enable-globalresources = `true|false`
Enables global resources[53]. 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[53] (and enables global resources)[53].

- globalresourcefile [gr]

  --gr | --globalresourcefile = *FILE*
  Specifies the global resource file[53] (and enables global resources)[53].

Common options

- 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 | unlimited*
  Specifies the error limit with a value range of `1` to `9999` or `unlimited`. The default value is `100`. Processing stops when the error limit is reached. Useful for limiting processor use during validation/transformation.

- info-limit

  --info-limit = *N | unlimited*
  Specifies the information message limit in the range `1-65535` or `unlimited`. Processing continues if the specified info limit is reached, but further messages are not reported. The default value is `100`.

- 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`.)

- 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.

- log-output

  --log-output = *FILE*

 © 2017-2023 Altova GmbH

Writes the log output to the specified file URL. Ensure that the CLI has write permission to the output location.

- network-timeout

  --network-timeout = *VALUE*
  Specifies the timeout in milliseconds for remote I/O operations. Default is: `40000`.

- recurse

  --recurse = `true|false`
  Used to select files within sub-directories, including in ZIP archives. 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. References to ZIP files must be given in quotes. The wildcard characters `*` and `?` may be used. So, `*.xml` will select all `.xml` files in the (zip) folder. The option's default value is `false`.
  Note: Boolean option values are set to `true` if the option is specified without a value.

- 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 Server. If used with a command, place `--version` before the command.

- warning-limit

  --warning-limit = *N | unlimited*
  Specifies the warning limit in the range `1-65535` or `unlimited`. Processing continues if this limit is reached, but further warnings are not reported. The default value is `100`.

5.2.3 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.

```
raptorxml 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

Examples of the **wfany** command:

- **raptorxml** wfany c:\Test.xml
- **raptorxml** wfany --error-format=text c:\Test.xml

▼ Casing and slashes on the command line

RaptorXML (and **RaptorXMLServer** for administration commands) *on Windows*
raptorxml (and **raptorxmlserver** for administration commands) *on Windows and Unix (Linux, Mac)*

* Note that lowercase (raptorxml and raptorxmlserver) works on all platforms (Windows, Linux, and Mac), while upper-lower (RaptorXML) works only on Windows and Mac.
* Use forward slashes on Linux and Mac, backslashes on Windows.

▼ Backslashes, spaces, and special characters on Windows systems

On Windows systems: When spaces or special characters occur in strings (for example in file or folder names, or company, person or product names), use quotes: for example, **"My File"**. Note, however, that a backslash followed by a double-quotation mark (for example, **"C:\My directory\"**) might not be read correctly. This is because the backslash character is also used to indicate the start of an escape sequence, and the escape sequence **\"** stands for the double-quotation mark character. If you want to escape this sequence of characters, use a preceding backslash, like this: **\\"**. To summarize: If you need to write a file path that contains spaces or an end backslash, write it like this: **"C:\My Directory\ \"**.

Options

Options are listed in short form (if available) and long form. You can use one or two dashes for both short and long forms. An option may or may not take a value. If it takes a value, it is written like this: **--option=value**. 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. If an option takes a Boolean value and no value is specified, then the option's default value is TRUE. Use the **--h, --help** option to display information about the command.

▼ 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.

 © 2017-2023 Altova GmbH

- 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\RaptorXMLServer2024\etc\RootCatalog.xml`). See the section, XML Catalogs [47], 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 [47], for information about working with catalogs.

  - enable-globalresources

    --enable-globalresources = `true|false`
    Enables global resources [53]. 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 [53] (and enables global resources) [53].

  - globalresourcefile [gr]

    --gr | --globalresourcefile = *FILE*
    Specifies the global resource file [53] (and enables global resources) [53].

- Common options
  - 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 | unlimited*
    Specifies the error limit with a value range of `1` to `9999` or `unlimited`. The default value is `100`. Processing stops when the error limit is reached. Useful for limiting processor use during validation/transformation.

  - info-limit

    --info-limit = *N | unlimited*
    Specifies the information message limit in the range `1-65535` or `unlimited`. Processing continues if the specified info limit is reached, but further messages are not reported. The default value is `100`.

  - 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`.)

- 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.

- log-output

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

- network-timeout

  --network-timeout = *VALUE*
  Specifies the timeout in milliseconds for remote I/O operations. Default is: `40000`.

- recurse

  --recurse = `true|false`
  Used to select files within sub-directories, including in ZIP archives. 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. References to ZIP files must be given in quotes. The wildcard characters `*` and `?` may be used. So, `*.xml` will select all `.xml` files in the (zip) folder. The option's default value is `false`.
  Note: Boolean option values are set to `true` if the option is specified without a value.

- 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 Server. If used with a command, place `--version` before the command.

- warning-limit

  --warning-limit = *N | unlimited*

 © 2017-2023 Altova GmbH

Specifies the warning limit in the range `1-65535` or `unlimited`. Processing continues if this limit is reached, but further warnings are not reported. The default value is `100`.

5.3 XQuery Commands

The XQuery commands are:

- xquery 92: for executing XQuery documents, optionally with an input document
- xqueryupdate 100: for executing an XQuery update, using an XQuery document and, optionally, the input XML document to update
- valxquery 108: for validating XQuery documents
- valxqueryupdate 114: for validating an XQuery (update) document

5.3.1 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.

```
raptorxml 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

Examples of the **xquery** command:

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

▼ Casing and slashes on the command line

RaptorXML (and **RaptorXMLServer** for administration commands) *on Windows*
raptorxml (and **raptorxmlserver** for administration commands) *on Windows and Unix (Linux, Mac)*

* Note that lowercase (raptorxml and raptorxmlserver) works on all platforms (Windows, Linux, and Mac), while upper-lower (RaptorXML) works only on Windows and Mac.
* Use forward slashes on Linux and Mac, backslashes on Windows.

▼ Backslashes, spaces, and special characters on Windows systems

On Windows systems: When spaces or special characters occur in strings (for example in file or folder names, or company, person or product names), use quotes: for example, **"My File"**. Note, however, that a backslash followed by a double-quotation mark (for example, **"C:\My directory\"**) might not be read correctly. This is because the backslash character is also used to indicate the start of an escape sequence, and the escape sequence **\"** stands for the double-quotation mark character. If you want to

 © 2017-2023 Altova GmbH

escape this sequence of characters, use a preceding backslash, like this: `\\"`. To summarize: If you need to write a file path that contains spaces or an end backslash, write it like this: `"C:\My Directory\ \"`.

Options

Options are listed in short form (if available) and long form. You can use one or two dashes for both short and long forms. An option may or may not take a value. If it takes a value, it is written like this: **`--option=value`**. 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. If an option takes a Boolean value and no value is specified, then the option's default value is `TRUE`. Use the **`--h, --help`** option to display information about the command.

- 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, xsltoutput

    `output =` *FILE*, **`xsltoutput =`** *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. Additional output files, such as generated image files, are reported as `xslt-additional-output-files`. If no `--output` or `--xsltoutput` 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;
    ```

    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:*

    ```
    raptorxml 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
    ```

- xpath-static-type-errors-as-warnings

  --xpath-static-type-errors-as-warnings = `true|false`
  If `true`, downgrades to warnings any type errors that are detected in the XPath static context. Whereas an error would cause the execution to fail, a warning would enable processing to continue. Default is `false`.

- xquery-version

  --xquery-version = `1|1.0|3|3.0|3.1`
  Specifies whether the XQuery processor should use XQuery 1.0 or XQuery 3.0. Default value is 3.1.

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

© 2017-2023 Altova GmbH

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

▼ 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 option specifies whether to load a schema document or just license a namespace, and, if a schema document is to be loaded, which information should be used to find it. Default: load-preferring-schemalocation.
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 47. 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 47. If no schemaLocation attribute is present, then the value of the namespace attribute is used via a catalog mapping 47. This is the **default value**.
- load-by-namespace: The value of the namespace attribute is used to locate the schema via a catalog mapping 47.
- load-combining-both: If either the namespace or schemaLocation attribute has a catalog mapping 47, then the mapping is used. If both have catalog mappings 47, then the value of the --schema-mapping option (XML/XSD option 225) decides which mapping is used. If no catalog mapping 47 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
Specifies the behavior of the xsi:schemaLocation and xsi:noNamespaceSchemaLocation attributes: Whether to load a schema document, and, if yes, which information should be used to find it. Default: load-by-schemalocation.

- The load-by-schemalocation value uses the URL of the schema location 383 in the xsi:schemaLocation and xsi:noNamespaceSchemaLocation attributes in XML instance documents. This is the **default value**.
- The load-by-namespace value takes the namespace part 383 of xsi:schemaLocation and an empty string in the case of xsi:noNamespaceSchemaLocation and locates the schema via a catalog mapping 47.
- If load-combining-both is used and if either the namespace part or the URL part has a catalog mapping 47, then the catalog mapping 47 is used. If both have catalog mappings 47, then the value of the --schema-mapping option (XML/XSD option 225) 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 schema location and namespace are both used to find a schema document, specifies which of them should be preferred during catalog lookup. (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[47], 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 for the XML instance document: `wf`=wellformed check; `id`=wellformed with ID/IDREF checks; `valid`=validation. Default value is `wf`. Note that a value of `valid` requires that each instance document loaded during processing references a DTD. If no DTD exists, an error is reported.

- xml-mode-for-schemas

  `--xml-mode-for-schemas =` `wf|id|valid`
  Specifies the XML processing mode to use for XML schema documents: `wf`=wellformed check; `id`=wellformed with ID/IDREF checks; `valid`=validation. Default value is `wf`. Note that a value of `valid` requires that each schema document loaded during processing references a DTD. If no DTD exists, an error is reported.

- xml-validation-error-as-warning

  `--xml-validation-error-as-warning =` `true|false`
  If `true`, treats validation errors as warnings. If errors are treated as warnings, additional processing, such as XSLT transformations, will continue regardless of errors. Default is `false`.

- xpath-static-type-errors-as-warnings

  `--xpath-static-type-errors-as-warnings =` `true|false`
  If `true`, downgrades to warnings any type errors that are detected in the XPath static context. Whereas an error would cause the execution to fail, a warning would enable processing to continue. Default is `false`.

- 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 more than one schema document.

- 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

 © 2017-2023 Altova GmbH

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\RaptorXMLServer2024\etc\RootCatalog.xml`). See the section, XML Catalogs [47], 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 [47], for information about working with catalogs.

enable-globalresources

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

Enables global resources [53]. 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 [53] (and enables global resources) [53].

globalresourcefile [gr]

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

Specifies the global resource file [53] (and enables global resources) [53].

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.

▼ jvm-location

--jvm-location = *FILE*
FILE specifies the location of the Java Virtual Machine (DLL on Windows, shared object on Linux). The JVM is needed if you use Java extension functions[485] in your XSLT/XQuery code. Default is `false`.

▼ javaext-barcode-location

--javaext-barcode-location = *FILE*
Specifies the path to the folder that contains the barcode extension file `AltovaBarcodeExtension.jar`. The path must be given in one of the following forms:

- A file URI, for example: `--javaext-barcode-location="file:///C:/Program Files/Altova/RaptorXMLServer2024/etc/jar/"`
- A Windows path with backslashes escaped, for example: `--javaext-barcode-location="C:\\Program Files\\Altova\\RaptorXMLServer2024\\etc\\jar\\"`

▼ 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.

▼ Common options

▼ 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 | unlimited*
Specifies the error limit with a value range of `1` to `9999` or `unlimited`. The default value is `100`. Processing stops when the error limit is reached. Useful for limiting processor use during validation/transformation.

▼ info-limit

--info-limit = *N | unlimited*
Specifies the information message limit in the range `1-65535` or `unlimited`. Processing continues if the specified info limit is reached, but further messages are not reported. The default value is `100`.

▼ help

--help
Displays help text for the command. For example, `valany --h`. (Alternatively the `help` command

 © 2017-2023 Altova GmbH

can be used with an argument. For example: `help valany`.)

- 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.

- log-output

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

- network-timeout

  `--network-timeout =` *`VALUE`*
  Specifies the timeout in milliseconds for remote I/O operations. Default is: `40000`.

- recurse

  `--recurse =` `true|false`
  Used to select files within sub-directories, including in ZIP archives. 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. References to ZIP files must be given in quotes. The wildcard characters `*` and `?` may be used. So, `*.xml` will select all `.xml` files in the (zip) folder. The option's default value is `false`.
  Note: Boolean option values are set to `true` if the option is specified without a value.

- 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 Server. If used with a command, place `--version` before the command.

- warning-limit

  `--warning-limit =` *`N | unlimited`*
  Specifies the warning limit in the range `1-65535` or `unlimited`. Processing continues if this limit is reached, but further warnings are not reported. The default value is `100`.

5.3.2 xqueryupdate

The **xqueryupdate** command takes an XQuery or XQuery Update file as its single argument and executes it. If an optional input XML file is specified, then this XML file is processed with the XQuery Update commands submitted in the ***XQuery(Update)-File***. In this case, the updates can be applied directly to the input file or the updated XML data can be written to an output XML file. The input and output files are specified as options. If the ***XQuery(Update)-File*** contains only XQuery instructions and no XQuery Update instructions, then the command carries out a straightforward XQuery execution.

```
raptorxml xqueryupdate [options] XQuery(Update)-File
```

- The argument ***XQuery(Update)-File*** is the path and name of the XQuery file (**.xq**) or XQuery Update (**.xqu**) file to be executed. If the file contains XQuery Update instructions, then these are executed on the input XML file. Otherwise, the command works as an XQuery execution command.
- You can specify whether XQuery Update 1.0 or 3.0 should be used. By default XQuery Update 3.0 is used.

Examples

Examples of the **xqueryupdate** command:

- **raptorxml** `xqueryupdate --output=c:\Output.xml c:\TestQuery.xq` (Writes the output of the XQuery file to the output file.)
- **raptorxml** `xqueryupdate --input=c:\Input.xml --output=c:\Output.xml --updated-xml=asmainresult c:\UpdateFile.xqu` (Updates `Input.xml` using the update instructions in `UpdateFile.xqu`, and writes the update to `Output.xml`.)
- **raptorxml** `xqueryupdate --input=c:\Input.xml --output=c:\Output.xml --updated-xml=writeback c:\UpdateFile.xq` (Updates `Input.xml` using the update instructions in `UpdateFile.xq`. The file `Output.xml` is not created.)
- **raptorxml** `xqueryupdate --input=c:\Input.xml --output=c:\Output.xml --updated-xml=discard c:\TestQuery.xqu` (Updates are discarded. The input file is not modified. The file `Output.xml` will be created, but will not contain any updated XML.)
- **raptorxml** `xqueryupdate --input=c:\Input.xml --output=c:\Output.xml c:\TestQuery.xqu` (Updates are discarded as in the previous example. This is because the default value of the `--updated-xml` option is `discard`.)

▼ Casing and slashes on the command line

RaptorXML (and **RaptorXMLServer** for administration commands) *on Windows*
raptorxml (and **raptorxmlserver** for administration commands) *on Windows and Unix (Linux, Mac)*

* Note that lowercase (`raptorxml` and `raptorxmlserver`) works on all platforms (Windows, Linux, and Mac), while upper-lower (`RaptorXML`) works only on Windows and Mac.
* Use forward slashes on Linux and Mac, backslashes on Windows.

 © 2017-2023 Altova GmbH

- Backslashes, spaces, and special characters on Windows systems

  On Windows systems: When spaces or special characters occur in strings (for example in file or folder names, or company, person or product names), use quotes: for example, `"My File"`. Note, however, that a backslash followed by a double-quotation mark (for example, `"C:\My directory\"`) might not be read correctly. This is because the backslash character is also used to indicate the start of an escape sequence, and the escape sequence `\"` stands for the double-quotation mark character. If you want to escape this sequence of characters, use a preceding backslash, like this: `\\"`. To summarize: If you need to write a file path that contains spaces or an end backslash, write it like this: `"C:\My Directory\ \"`.

Options

Options are listed in short form (if available) and long form. You can use one or two dashes for both short and long forms. An option may or may not take a value. If it takes a value, it is written like this: `--option=value`. 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. If an option takes a Boolean value and no value is specified, then the option's default value is `TRUE`. Use the `--h, --help` option to display information about the command.

- 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, xsltoutput

    `output = FILE, xsltoutput = 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. Additional output files, such as generated image files, are reported as `xslt-additional-output-files`. If no `--output` or `--xsltoutput` 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;
    ```

    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:*

    ```
    raptorxml 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
    ```

- xpath-static-type-errors-as-warnings

  `--xpath-static-type-errors-as-warnings =` `true|false`
  If `true`, downgrades to warnings any type errors that are detected in the XPath static context. Whereas an error would cause the execution to fail, a warning would enable processing to continue. Default is `false`.

© 2017-2023 Altova GmbH

- xquery-update-version

  --xquery-update-version = 1|1.0|3|3.0|
  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.

  - discard: The update is discarded and not written to file. Neither the input file nor the output file will be updated. Note that this is the default.
  - writeback: Writes the update back to the input XML file that is specified with the --input option.
  - asmainresult: Writes the update to the output XML file that is specified with the --output option. If the **--output** option is not specified, then the update is written to the standard output. In both cases, the input XML file will not be modified.

  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: true.

  - 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 option specifies whether to load a schema document or just license a namespace, and, if a schema document is to be loaded, which information should be used to find it. Default: load-preferring-schemalocation.
    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 [47]. 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 [47]. If no schemaLocation attribute is present, then the value of the namespace attribute is used via a catalog mapping [47]. This is the **default value**.
    - load-by-namespace: The value of the namespace attribute is used to locate the schema via a catalog mapping [47].

- `load-combining-both`: If either the `namespace` or `schemaLocation` attribute has a catalog mapping[47], then the mapping is used. If both have catalog mappings[47], then the value of the `--schema-mapping` option (XML/XSD option[225]) decides which mapping is used. If no catalog mapping[47] 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`
Specifies the behavior of the `xsi:schemaLocation` and `xsi:noNamespaceSchemaLocation` attributes: Whether to load a schema document, and, if yes, which information should be used to find it. Default: `load-by-schemalocation`.

- The `load-by-schemalocation` value uses the URL of the schema location[383] in the `xsi:schemaLocation` and `xsi:noNamespaceSchemaLocation` attributes in XML instance documents. This is the **default value**.
- The `load-by-namespace` value takes the namespace part[383] of `xsi:schemaLocation` and an empty string in the case of `xsi:noNamespaceSchemaLocation` and locates the schema via a catalog mapping[47].
- If `load-combining-both` is used and if either the namespace part or the URL part has a catalog mapping[47], then the catalog mapping[47] is used. If both have catalog mappings[47], then the value of the --schema-mapping option (XML/XSD option[225]) 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 schema location and namespace are both used to find a schema document, specifies which of them should be preferred during catalog lookup. (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[47], 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 for the XML instance document: `wf`=wellformed check; `id`=wellformed with ID/IDREF checks; `valid`=validation. Default value is `wf`. Note that a value of `valid` requires that each instance document loaded during processing references a DTD. If no DTD exists, an error is reported.

© 2017-2023 Altova GmbH

- xml-mode-for-schemas

  --xml-mode-for-schemas = `wf|id|valid`
  Specifies the XML processing mode to use for XML schema documents: `wf`=wellformed check; `id`=wellformed with ID/IDREF checks; `valid`=validation. Default value is `wf`. Note that a value of `valid` requires that each schema document loaded during processing references a DTD. If no DTD exists, an error is reported.

- xml-validation-error-as-warning

  --xml-validation-error-as-warning = `true|false`
  If `true`, treats validation errors as warnings. If errors are treated as warnings, additional processing, such as XSLT transformations, will continue regardless of errors. Default is `false`.

- 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 more than one schema document.

- 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\RaptorXMLServer2024\etc\RootCatalog.xml`). See the section, XML Catalogs 47, 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 47, for information about working with catalogs.

  - enable-globalresources

    --enable-globalresources = `true|false`
    Enables global resources 53. 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[53] (and enables global resources)[53].

- globalresourcefile [gr]

  --gr | --globalresourcefile = *FILE*
  Specifies the global resource file[53] (and enables global resources)[53].

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.

- jvm-location

  --jvm-location = *FILE*
  FILE specifies the location of the Java Virtual Machine (DLL on Windows, shared object on Linux). The JVM is needed if you use Java extension functions[485] in your XSLT/XQuery code. Default is `false`.

- javaext-barcode-location

  --javaext-barcode-location = *FILE*
  Specifies the path to the folder that contains the barcode extension file `AltovaBarcodeExtension.jar`. The path must be given in one of the following forms:

  - A file URI, for example: `--javaext-barcode-location="file:///C:/Program Files/Altova/RaptorXMLServer2024/etc/jar/"`
  - A Windows path with backslashes escaped, for example: `--javaext-barcode-location="C:\\Program Files\\Altova\\RaptorXMLServer2024\\etc\\jar\\"`

- 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.

Common options

 © 2017-2023 Altova GmbH

- 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 | unlimited*
  Specifies the error limit with a value range of `1` to `9999` or `unlimited`. The default value is `100`. Processing stops when the error limit is reached. Useful for limiting processor use during validation/transformation.

- info-limit

  --info-limit = *N | unlimited*
  Specifies the information message limit in the range `1-65535` or `unlimited`. Processing continues if the specified info limit is reached, but further messages are not reported. The default value is `100`.

- 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`.)

- 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.

- log-output

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

- network-timeout

  --network-timeout = *VALUE*
  Specifies the timeout in milliseconds for remote I/O operations. Default is: `40000`.

- recurse

  --recurse = `true|false`
  Used to select files within sub-directories, including in ZIP archives. 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. References to ZIP files must be given in quotes. The wildcard characters `*` and `?` may be used. So, `*.xml` will select all `.xml` files in the (zip) folder. The option's default value is `false`.
  Note: Boolean option values are set to `true` if the option is specified without a value.

- 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 Server. If used with a command, place `--version` before the command.

- warning-limit

  --warning-limit = *N | unlimited*
  Specifies the warning limit in the range `1-65535` or `unlimited`. Processing continues if this limit is reached, but further warnings are not reported. The default value is `100`.

5.3.3 valxquery

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

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

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

Examples

Examples of the **valxquery** command:

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

- Casing and slashes on the command line

  RaptorXML (and **RaptorXMLServer** for administration commands) *on Windows*
  raptorxml (and **raptorxmlserver** for administration commands) *on Windows and Unix (Linux, Mac)*

  * Note that lowercase (`raptorxml` and `raptorxmlserver`) works on all platforms (Windows, Linux, and Mac), while upper-lower (`RaptorXML`) works only on Windows and Mac.
  * Use forward slashes on Linux and Mac, backslashes on Windows.

 © 2017-2023 Altova GmbH

- Backslashes, spaces, and special characters on Windows systems

  On Windows systems: When spaces or special characters occur in strings (for example in file or folder names, or company, person or product names), use quotes: for example, `"My File"`. Note, however, that a backslash followed by a double-quotation mark (for example, `"C:\My directory\"`) might not be read correctly. This is because the backslash character is also used to indicate the start of an escape sequence, and the escape sequence `\"` stands for the double-quotation mark character. If you want to escape this sequence of characters, use a preceding backslash, like this: `\\"`. To summarize: If you need to write a file path that contains spaces or an end backslash, write it like this: `"C:\My Directory\ \"`.

Options

Options are listed in short form (if available) and long form. You can use one or two dashes for both short and long forms. An option may or may not take a value. If it takes a value, it is written like this: `--option=value`. 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. If an option takes a Boolean value and no value is specified, then the option's default value is `TRUE`. Use the `--h, --help` option to display information about the command.

- 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|1.0|3|3.0|3.1`
    Specifies whether the XQuery processor should use XQuery 1.0 or XQuery 3.0. Default value is 3.1.

- 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: `true`.

  - 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 option specifies whether to load a schema document or just

license a namespace, and, if a schema document is to be loaded, which information should be used to find it. Default: `load-preferring-schemalocation`.
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[47]. 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[47]. If no `schemaLocation` attribute is present, then the value of the `namespace` attribute is used via a catalog mapping[47]. This is the **default value**.
- `load-by-namespace`: The value of the `namespace` attribute is used to locate the schema via a catalog mapping[47].
- `load-combining-both`: If either the `namespace` or `schemaLocation` attribute has a catalog mapping[47], then the mapping is used. If both have catalog mappings[47], then the value of the `--schema-mapping` option (XML/XSD option[225]) decides which mapping is used. If no catalog mapping[47] 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`
Specifies the behavior of the `xsi:schemaLocation` and `xsi:noNamespaceSchemaLocation` attributes: Whether to load a schema document, and, if yes, which information should be used to find it. Default: `load-by-schemalocation`.

- The `load-by-schemalocation` value uses the URL of the schema location[383] in the `xsi:schemaLocation` and `xsi:noNamespaceSchemaLocation` attributes in XML instance documents. This is the **default value**.
- The `load-by-namespace` value takes the namespace part[383] of `xsi:schemaLocation` and an empty string in the case of `xsi:noNamespaceSchemaLocation` and locates the schema via a catalog mapping[47].
- If `load-combining-both` is used and if either the namespace part or the URL part has a catalog mapping[47], then the catalog mapping[47] is used. If both have catalog mappings[47], then the value of the --schema-mapping option (XML/XSD option[225]) 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 schema location and namespace are both used to find a schema document, specifies which of them should be preferred during catalog lookup. (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[47], 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`

 © 2017-2023 Altova GmbH

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 for the XML instance document: `wf`=wellformed check; `id`=wellformed with ID/IDREF checks; `valid`=validation. Default value is `wf`. Note that a value of `valid` requires that each instance document loaded during processing references a DTD. If no DTD exists, an error is reported.

- xml-mode-for-schemas

  `--xml-mode-for-schemas =` `wf|id|valid`
  Specifies the XML processing mode to use for XML schema documents: `wf`=wellformed check; `id`=wellformed with ID/IDREF checks; `valid`=validation. Default value is `wf`. Note that a value of `valid` requires that each schema document loaded during processing references a DTD. If no DTD exists, an error is reported.

- xpath-static-type-errors-as-warnings

  `--xpath-static-type-errors-as-warnings =` `true|false`
  If `true`, downgrades to warnings any type errors that are detected in the XPath static context. Whereas an error would cause the execution to fail, a warning would enable processing to continue. Default is `false`.

- 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\RaptorXMLServer2024\etc\RootCatalog.xml`). See the section, XML Catalogs 47, 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 [47], for information about working with catalogs.

- enable-globalresources

  --enable-globalresources = `true|false`
  Enables global resources [53]. 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 [53] (and enables global resources) [53].

- globalresourcefile [gr]

  --gr | --globalresourcefile = *FILE*
  Specifies the global resource file [53] (and enables global resources) [53].

- 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.

  - jvm-location

    --jvm-location = *FILE*
    FILE specifies the location of the Java Virtual Machine (DLL on Windows, shared object on Linux). The JVM is needed if you use Java extension functions [485] in your XSLT/XQuery code. Default is `false`.

  - javaext-barcode-location

    --javaext-barcode-location = *FILE*
    Specifies the path to the folder that contains the barcode extension file `AltovaBarcodeExtension.jar`. The path must be given in one of the following forms:

    - A file URI, for example: `--javaext-barcode-location="file:///C:/Program Files/Altova/RaptorXMLServer2024/etc/jar/"`
    - A Windows path with backslashes escaped, for example: `--javaext-barcode-location="C:\\Program Files\\Altova\\RaptorXMLServer2024\\etc\\jar\\"`

 © 2017-2023 Altova GmbH

- 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.

- Common options
  - 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 | unlimited*
    Specifies the error limit with a value range of `1` to `9999` or `unlimited`. The default value is `100`. Processing stops when the error limit is reached. Useful for limiting processor use during validation/transformation.

  - info-limit

    --info-limit = *N | unlimited*
    Specifies the information message limit in the range `1-65535` or `unlimited`. Processing continues if the specified info limit is reached, but further messages are not reported. The default value is `100`.

  - 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`.)

  - 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.

  - log-output

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

  - network-timeout

    --network-timeout = *VALUE*
    Specifies the timeout in milliseconds for remote I/O operations. Default is: `40000`.

- recurse

  --recurse = `true|false`
  Used to select files within sub-directories, including in ZIP archives. 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. References to ZIP files must be given in quotes. The wildcard characters `*` and `?` may be used. So, `*.xml` will select all `.xml` files in the (zip) folder. The option's default value is `false`.
  Note: Boolean option values are set to `true` if the option is specified without a value.

- 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 Server. If used with a command, place `--version` before the command.

- warning-limit

  --warning-limit = *N | unlimited*
  Specifies the warning limit in the range `1-65535` or `unlimited`. Processing continues if this limit is reached, but further warnings are not reported. The default value is `100`.

5.3.4 valxqueryupdate

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

```
raptorxml valxqueryupdate [options] XQuery-File
```

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

Examples

Examples of the **valxqueryupdate** command:

- **raptorxml** valxqueryupdate c:\Test.xqu
- **raptorxml** valxqueryupdate --xquery-update-version=1 c:\Test.xqu

© 2017-2023 Altova GmbH

- Casing and slashes on the command line

  RaptorXML (and **RaptorXMLServer** for administration commands) *on Windows*
  raptorxml (and **raptorxmlserver** for administration commands) *on Windows and Unix (Linux, Mac)*

  * Note that lowercase (raptorxml and raptorxmlserver) works on all platforms (Windows, Linux, and Mac), while upper-lower (RaptorXML) works only on Windows and Mac.
  * Use forward slashes on Linux and Mac, backslashes on Windows.

- Backslashes, spaces, and special characters on Windows systems

  On Windows systems: When spaces or special characters occur in strings (for example in file or folder names, or company, person or product names), use quotes: for example, **"My File"**. Note, however, that a backslash followed by a double-quotation mark (for example, **"C:\My directory\"**) might not be read correctly. This is because the backslash character is also used to indicate the start of an escape sequence, and the escape sequence **\"** stands for the double-quotation mark character. If you want to escape this sequence of characters, use a preceding backslash, like this: **\\"**. To summarize: If you need to write a file path that contains spaces or an end backslash, write it like this: **"C:\My Directory\ \"**.

Options

Options are listed in short form (if available) and long form. You can use one or two dashes for both short and long forms. An option may or may not take a value. If it takes a value, it is written like this: **--option=value**. 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. If an option takes a Boolean value and no value is specified, then the option's default value is TRUE. Use the **--h, --help** option to display information about the command.

- 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|1.0|3|3.0|
    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: true.

▼ 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 option specifies whether to load a schema document or just license a namespace, and, if a schema document is to be loaded, which information should be used to find it. Default: load-preferring-schemalocation.
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[47]. 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[47]. If no schemaLocation attribute is present, then the value of the namespace attribute is used via a catalog mapping[47]. This is the **default value**.
- load-by-namespace: The value of the namespace attribute is used to locate the schema via a catalog mapping[47].
- load-combining-both: If either the namespace or schemaLocation attribute has a catalog mapping[47], then the mapping is used. If both have catalog mappings[47], then the value of the --schema-mapping option (XML/XSD option[225]) decides which mapping is used. If no catalog mapping[47] 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
Specifies the behavior of the xsi:schemaLocation and xsi:noNamespaceSchemaLocation attributes: Whether to load a schema document, and, if yes, which information should be used to find it. Default: load-by-schemalocation.

- The load-by-schemalocation value uses the URL of the schema location[383] in the xsi:schemaLocation and xsi:noNamespaceSchemaLocation attributes in XML instance documents. This is the **default value**.
- The load-by-namespace value takes the namespace part[383] of xsi:schemaLocation and an empty string in the case of xsi:noNamespaceSchemaLocation and locates the schema via a catalog mapping[47].
- If load-combining-both is used and if either the namespace part or the URL part has a catalog mapping[47], then the catalog mapping[47] is used. If both have catalog mappings[47], then the value of the --schema-mapping option (XML/XSD option[225]) 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 schema location and namespace are both used to find a schema document, specifies which of them should be preferred during catalog lookup. (If either the --schemalocation-hints or the --

 © 2017-2023 Altova GmbH

`schema-imports` option has a value of `load-combining-both`, and if the namespace and URL parts involved both have catalog mappings[47], 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 for the XML instance document: `wf`=wellformed check; `id`=wellformed with ID/IDREF checks; `valid`=validation. Default value is `wf`. Note that a value of `valid` requires that each instance document loaded during processing references a DTD. If no DTD exists, an error is reported.

- xml-mode-for-schemas

  `--xml-mode-for-schemas =` `wf|id|valid`
  Specifies the XML processing mode to use for XML schema documents: `wf`=wellformed check; `id`=wellformed with ID/IDREF checks; `valid`=validation. Default value is `wf`. Note that a value of `valid` requires that each schema document loaded during processing references a DTD. If no DTD exists, an error is reported.

- xpath-static-type-errors-as-warnings

  `--xpath-static-type-errors-as-warnings =` `true|false`
  If `true`, downgrades to warnings any type errors that are detected in the XPath static context. Whereas an error would cause the execution to fail, a warning would enable processing to continue. Default is `false`.

- 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\RaptorXMLServer2024\etc\RootCatalog.xml`). See the section, XML Catalogs[47], 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[47], for information about working with catalogs.

- enable-globalresources

  `--enable-globalresources =` `true|false`
  Enables global resources[53]. 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[53] (and enables global resources)[53].

- globalresourcefile [gr]

  `--gr | --globalresourcefile =` *FILE*
  Specifies the global resource file[53] (and enables global resources)[53].

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.

- jvm-location

  `--jvm-location =` *FILE*
  FILE specifies the location of the Java Virtual Machine (DLL on Windows, shared object on Linux). The JVM is needed if you use Java extension functions[485] in your XSLT/XQuery code. Default is `false`.

- javaext-barcode-location

  `--javaext-barcode-location =` *FILE*

 © 2017-2023 Altova GmbH

Specifies the path to the folder that contains the barcode extension file `AltovaBarcodeExtension.jar`. The path must be given in one of the following forms:

- A file URI, for example: `--javaext-barcode-location="file:///C:/Program Files/Altova/RaptorXMLServer2024/etc/jar/"`
- A Windows path with backslashes escaped, for example: `--javaext-barcode-location="C:\\Program Files\\Altova\\RaptorXMLServer2024\\etc\\jar\\"`

▼ 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.

▼ Common options

▼ 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 | unlimited`*
Specifies the error limit with a value range of `1` to `9999` or `unlimited`. The default value is `100`. Processing stops when the error limit is reached. Useful for limiting processor use during validation/transformation.

▼ info-limit

`--info-limit =` *`N | unlimited`*
Specifies the information message limit in the range `1-65535` or `unlimited`. Processing continues if the specified info limit is reached, but further messages are not reported. The default value is `100`.

▼ 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`.)

▼ 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.

▼ log-output

`--log-output =` *`FILE`*

Writes the log output to the specified file URL. Ensure that the CLI has write permission to the output location.

- network-timeout

  --network-timeout = *VALUE*
  Specifies the timeout in milliseconds for remote I/O operations. Default is: `40000`.

- recurse

  --recurse = `true|false`
  Used to select files within sub-directories, including in ZIP archives. 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. References to ZIP files must be given in quotes. The wildcard characters `*` and `?` may be used. So, `*.xml` will select all `.xml` files in the (zip) folder. The option's default value is `false`.
  Note: Boolean option values are set to `true` if the option is specified without a value.

- 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 Server. If used with a command, place `--version` before the command.

- warning-limit

  --warning-limit = *N | unlimited*
  Specifies the warning limit in the range `1-65535` or `unlimited`. Processing continues if this limit is reached, but further warnings are not reported. The default value is `100`.

© 2017-2023 Altova GmbH

5.4 XSLT Commands

The XSLT commands are:

- xslt[121]: for transforming XML documents with an XSLT document
- valxslt[129]: for validating XSLT documents

5.4.1 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[229].

```
raptorxml 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[229]) or a named template entry point (--template-entry-point[229]) is required.
- To transform JSON data, load the JSON data via the **json-doc($path)** function of XPath 3.1, and use the **xslt** command's **--initial-match-selection** option. See the last item in the examples given below.
- If no --output[229] 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

Examples of the **xslt** command:

- **raptorxml** xslt --input=c:\Test.xml --output=c:\Output.xml c:\Test.xslt
- **raptorxml** xslt --template-entry-point=StartTemplate --output=c:\Output.xml c:\Test.xslt
- **raptorxml** 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
- **raptorxml** xslt --initial-match-selection=json-doc('MyData.json',**map**{'liberal':true()}) --output=c:\MyData.xml c:\Test.xslt
- **raptorxml** xslt --initial-match-selection="**json-doc(**'MyData.json',**map**{'liberal':true()}**)**" --output=c:\MyData.xml c:\Test.xslt (If the json-doc argument string contains spaces, then enclose the entire json-doc value in quotes.)

▼ Casing and slashes on the command line

RaptorXML (and **RaptorXMLServer** for administration commands) *on Windows*
raptorxml (and **raptorxmlserver** for administration commands) *on Windows and Unix (Linux, Mac)*

* Note that lowercase (raptorxml and raptorxmlserver) works on all platforms (Windows, Linux, and Mac), while upper-lower (RaptorXML) works only on Windows and Mac.
* Use forward slashes on Linux and Mac, backslashes on Windows.

- Backslashes, spaces, and special characters on Windows systems

  On Windows systems: When spaces or special characters occur in strings (for example in file or folder names, or company, person or product names), use quotes: for example, `"My File"`. Note, however, that a backslash followed by a double-quotation mark (for example, `"C:\My directory\"`) might not be read correctly. This is because the backslash character is also used to indicate the start of an escape sequence, and the escape sequence `\"` stands for the double-quotation mark character. If you want to escape this sequence of characters, use a preceding backslash, like this: `\\"`. To summarize: If you need to write a file path that contains spaces or an end backslash, write it like this: `"C:\My Directory\ \"`.

Options

Options are listed in short form (if available) and long form. You can use one or two dashes for both short and long forms. An option may or may not take a value. If it takes a value, it is written like this: `--option=value`. 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. If an option takes a Boolean value and no value is specified, then the option's default value is `TRUE`. Use the `--h, --help` option to display information about the command.

- XSLT processing
  - indent-characters

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

  - function-param

    `--function-param = VALUE`
    Specifies the functions that will be passed to the initial function. To specify more than one function, use the option multiple times. Note, however, that order is important.

  - global-context-item

    `--global-context-item = VALUE`
    Specifies the context item that is to be used to evaluate global variables.

  - initial-function

    `--initial-function = VALUE`
    The name of a function that is to be executed as the entry point of the transformation.

  - initial-match-selection

    `--initial-match-selection = VALUE`
    Specifies the value (sequence) of the initial match selection.

  - initial-mode, template-mode

    `--initial-mode, --template-mode = VALUE`
    Specifies the template mode to use for the transformation.

 © 2017-2023 Altova GmbH

- initial-template, template-entry-point

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

- input

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

- output, xsltoutput

  output = *FILE*, **xsltoutput =** *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. Additional output files, such as generated image files, are reported as `xslt-additional-output-files`. If no `--output` or `--xsltoutput` 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;
    ```

    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:*

    ```
    raptorxml 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-serialization-enabled

  --streaming-serialization-enabled = `true|false`

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

- template-param

  `--template-param =` *KEY:VALUE*
  Specifies parameters that will be passed to the initial template only (and not to any descending template call). To specify multiple parameters, use the option once for each parameter.

- tunnel-param

  `--tunnel-param =` *KEY:VALUE*
  Specifies parameters that will be passed to the initial template and to descending template calls. To specify multiple parameters, use the option once for each parameter.

- xpath-static-type-errors-as-warnings

  `--xpath-static-type-errors-as-warnings =` `true|false`
  If `true`, downgrades to warnings any type errors that are detected in the XPath static context. Whereas an error would cause the execution to fail, a warning would enable processing to continue. Default is `false`.

- xslt-version

  `--xslt-version =` `1|1.0|2|2.0|3|3.0|3.1`
  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: `true`.

  - 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 option specifies whether to load a schema document or just license a namespace, and, if a schema document is to be loaded, which information should be used to find it. Default: `load-preferring-schemalocation`.
    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 [47]. 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 [47]. If no `schemaLocation` attribute is present, then the

 © 2017-2023 Altova GmbH

value of the `namespace` attribute is used via a catalog mapping[47]. This is the **default value**.
- `load-by-namespace`: The value of the `namespace` attribute is used to locate the schema via a catalog mapping[47].
- `load-combining-both`: If either the `namespace` or `schemaLocation` attribute has a catalog mapping[47], then the mapping is used. If both have catalog mappings[47], then the value of the `--schema-mapping` option (XML/XSD option[225]) decides which mapping is used. If no catalog mapping[47] 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`
Specifies the behavior of the `xsi:schemaLocation` and `xsi:noNamespaceSchemaLocation` attributes: Whether to load a schema document, and, if yes, which information should be used to find it. Default: `load-by-schemalocation`.

- The `load-by-schemalocation` value uses the URL of the schema location[383] in the `xsi:schemaLocation` and `xsi:noNamespaceSchemaLocation` attributes in XML instance documents. This is the **default value**.
- The `load-by-namespace` value takes the namespace part[383] of `xsi:schemaLocation` and an empty string in the case of `xsi:noNamespaceSchemaLocation` and locates the schema via a catalog mapping[47].
- If `load-combining-both` is used and if either the namespace part or the URL part has a catalog mapping[47], then the catalog mapping[47] is used. If both have catalog mappings[47], then the value of the --schema-mapping option (XML/XSD option[225]) 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 schema location and namespace are both used to find a schema document, specifies which of them should be preferred during catalog lookup. (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[47], 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 for the XML instance document: `wf`=wellformed check; `id`=wellformed with ID/IDREF checks; `valid`=validation. Default value is `wf`. Note that a value of `valid` requires that each instance document loaded during processing references a DTD. If no DTD

exists, an error is reported.

- xml-mode-for-schemas

  --xml-mode-for-schemas = wf|id|valid
  Specifies the XML processing mode to use for XML schema documents: wf=wellformed check; id=wellformed with ID/IDREF checks; valid=validation. Default value is wf. Note that a value of valid requires that each schema document loaded during processing references a DTD. If no DTD exists, an error is reported.

- xml-validation-error-as-warning

  --xml-validation-error-as-warning = true|false
  If true, treats validation errors as warnings. If errors are treated as warnings, additional processing, such as XSLT transformations, will continue regardless of errors. Default is false.

- 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 more than one schema document.

- 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\RaptorXMLServer2024\etc\RootCatalog.xml). See the section, XML Catalogs 47, 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 47, for information about working with catalogs.

  - enable-globalresources

    --enable-globalresources = true|false
    Enables global resources 53. Default value is false.

 © 2017-2023 Altova GmbH

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 [53] (and enables global resources) [53].

- globalresourcefile [gr]

  `--gr | --globalresourcefile =` *FILE*
  Specifies the global resource file [53] (and enables global resources) [53].

- 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.

  - jvm-location

    `--jvm-location =` *FILE*
    FILE specifies the location of the Java Virtual Machine (DLL on Windows, shared object on Linux). The JVM is needed if you use Java extension functions [485] in your XSLT/XQuery code. Default is `false`.

  - javaext-barcode-location

    `--javaext-barcode-location =` *FILE*
    Specifies the path to the folder that contains the barcode extension file `AltovaBarcodeExtension.jar`. The path must be given in one of the following forms:

    - A file URI, for example: `--javaext-barcode-location="file:///C:/Program Files/Altova/RaptorXMLServer2024/etc/jar/"`
    - A Windows path with backslashes escaped, for example: `--javaext-barcode-location="C:\\Program Files\\Altova\\RaptorXMLServer2024\\etc\\jar\\"`

  - 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.

- Common options
  - 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 | unlimited*
    Specifies the error limit with a value range of 1 to 9999 or unlimited. The default value is 100. Processing stops when the error limit is reached. Useful for limiting processor use during validation/transformation.

  - info-limit

    --info-limit = *N | unlimited*
    Specifies the information message limit in the range 1-65535 or unlimited. Processing continues if the specified info limit is reached, but further messages are not reported. The default value is 100.

  - 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.)

  - 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.

  - log-output

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

  - network-timeout

    --network-timeout = *VALUE*
    Specifies the timeout in milliseconds for remote I/O operations. Default is: 40000.

  - recurse

    --recurse = true|false
    Used to select files within sub-directories, including in ZIP archives. 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. References to

 © 2017-2023 Altova GmbH

ZIP files must be given in quotes. The wildcard characters `*` and `?` may be used. So, `*.xml` will select all `.xml` files in the (zip) folder. The option's default value is `false`.
Note: Boolean option values are set to `true` if the option is specified without a value.

- 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 Server. If used with a command, place `--version` before the command.

- warning-limit

  `--warning-limit =` *`N | unlimited`*
  Specifies the warning limit in the range `1-65535` or `unlimited`. Processing continues if this limit is reached, but further warnings are not reported. The default value is `100`.

5.4.2 valxslt

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

```
raptorxml 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

Examples of the **`valxslt`** command:

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

- Casing and slashes on the command line

  `RaptorXML` (and **`RaptorXMLServer`** for administration commands) *on Windows*

raptorxml (and **raptorxmlserver** for administration commands) *on Windows and Unix (Linux, Mac)*

* Note that lowercase (raptorxml and raptorxmlserver) works on all platforms (Windows, Linux, and Mac), while upper-lower (RaptorXML) works only on Windows and Mac.
* Use forward slashes on Linux and Mac, backslashes on Windows.

▼ Backslashes, spaces, and special characters on Windows systems

On Windows systems: When spaces or special characters occur in strings (for example in file or folder names, or company, person or product names), use quotes: for example, **"My File"**. Note, however, that a backslash followed by a double-quotation mark (for example, **"C:\My directory\"**) might not be read correctly. This is because the backslash character is also used to indicate the start of an escape sequence, and the escape sequence **\"** stands for the double-quotation mark character. If you want to escape this sequence of characters, use a preceding backslash, like this: **\\"**. To summarize: If you need to write a file path that contains spaces or an end backslash, write it like this: **"C:\My Directory\ \"**.

Options

Options are listed in short form (if available) and long form. You can use one or two dashes for both short and long forms. An option may or may not take a value. If it takes a value, it is written like this: **--option=value**. 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. If an option takes a Boolean value and no value is specified, then the option's default value is TRUE. Use the **--h, --help** option to display information about the command.

▼ XSLT processing

▼ initial-mode, template-mode

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

▼ initial-template, template-entry-point

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

▼ xslt-version

--xslt-version = 1|1.0|2|2.0|3|3.0|3.1
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.

 © 2017-2023 Altova GmbH

Default is: `true`.

- 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 option specifies whether to load a schema document or just license a namespace, and, if a schema document is to be loaded, which information should be used to find it. Default: `load-preferring-schemalocation`.
  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 [47]. 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 [47]. If no `schemaLocation` attribute is present, then the value of the `namespace` attribute is used via a catalog mapping [47]. This is the **default value**.
  - `load-by-namespace`: The value of the `namespace` attribute is used to locate the schema via a catalog mapping [47].
  - `load-combining-both`: If either the `namespace` or `schemaLocation` attribute has a catalog mapping [47], then the mapping is used. If both have catalog mappings [47], then the value of the `--schema-mapping` option (XML/XSD option [225]) decides which mapping is used. If no catalog mapping [47] 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`
  Specifies the behavior of the `xsi:schemaLocation` and `xsi:noNamespaceSchemaLocation` attributes: Whether to load a schema document, and, if yes, which information should be used to find it. Default: `load-by-schemalocation`.

  - The `load-by-schemalocation` value uses the URL of the schema location [383] in the `xsi:schemaLocation` and `xsi:noNamespaceSchemaLocation` attributes in XML instance documents. This is the **default value**.
  - The `load-by-namespace` value takes the namespace part [383] of `xsi:schemaLocation` and an empty string in the case of `xsi:noNamespaceSchemaLocation` and locates the schema via a catalog mapping [47].
  - If `load-combining-both` is used and if either the namespace part or the URL part has a catalog mapping [47], then the catalog mapping [47] is used. If both have catalog mappings [47], then the value of the --schema-mapping option (XML/XSD option [225]) 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 schema location and namespace are both used to find a schema document, specifies which of

them should be preferred during catalog lookup. (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[47], 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 for the XML instance document: `wf`=wellformed check; `id`=wellformed with ID/IDREF checks; `valid`=validation. Default value is `wf`. Note that a value of `valid` requires that each instance document loaded during processing references a DTD. If no DTD exists, an error is reported.

- xml-mode-for-schemas

  `--xml-mode-for-schemas =` `wf|id|valid`
  Specifies the XML processing mode to use for XML schema documents: `wf`=wellformed check; `id`=wellformed with ID/IDREF checks; `valid`=validation. Default value is `wf`. Note that a value of `valid` requires that each schema document loaded during processing references a DTD. If no DTD exists, an error is reported.

- xpath-static-type-errors-as-warnings

  `--xpath-static-type-errors-as-warnings =` `true|false`
  If `true`, downgrades to warnings any type errors that are detected in the XPath static context. Whereas an error would cause the execution to fail, a warning would enable processing to continue. Default is `false`.

- 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`*

 © 2017-2023 Altova GmbH

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\RaptorXMLServer2024\etc\RootCatalog.xml`). See the section, XML Catalogs [47], 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 [47], for information about working with catalogs.

- enable-globalresources

  `--enable-globalresources =` `true|false`
  Enables global resources [53]. 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 [53] (and enables global resources) [53].

- globalresourcefile [gr]

  `--gr | --globalresourcefile =` *FILE*
  Specifies the global resource file [53] (and enables global resources) [53].

- 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.

  - jvm-location

    `--jvm-location =` *FILE*
    FILE specifies the location of the Java Virtual Machine (DLL on Windows, shared object on Linux). The JVM is needed if you use Java extension functions [485] in your XSLT/XQuery code. Default is `false`.

  - javaext-barcode-location

--javaext-barcode-location = *FILE*
Specifies the path to the folder that contains the barcode extension file `AltovaBarcodeExtension.jar`. The path must be given in one of the following forms:

- A file URI, for example: `--javaext-barcode-location="file:///C:/Program Files/Altova/RaptorXMLServer2024/etc/jar/"`
- A Windows path with backslashes escaped, for example: `--javaext-barcode-location="C:\\Program Files\\Altova\\RaptorXMLServer2024\\etc\\jar\\"`

▼ 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.

▼ Common options

▼ 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 | unlimited*
Specifies the error limit with a value range of `1` to `9999` or `unlimited`. The default value is `100`. Processing stops when the error limit is reached. Useful for limiting processor use during validation/transformation.

▼ info-limit

--info-limit = *N | unlimited*
Specifies the information message limit in the range `1-65535` or `unlimited`. Processing continues if the specified info limit is reached, but further messages are not reported. The default value is `100`.

▼ 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`.)

▼ 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.

▼ log-output

 © 2017-2023 Altova GmbH

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

- network-timeout

  --network-timeout = *VALUE*
  Specifies the timeout in milliseconds for remote I/O operations. Default is: `40000`.

- recurse

  --recurse = `true|false`
  Used to select files within sub-directories, including in ZIP archives. 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. References to ZIP files must be given in quotes. The wildcard characters `*` and `?` may be used. So, `*.xml` will select all `.xml` files in the (zip) folder. The option's default value is `false`.
  Note: Boolean option values are set to `true` if the option is specified without a value.

- 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 Server. If used with a command, place `--version` before the command.

- warning-limit

  --warning-limit = *N | unlimited*
  Specifies the warning limit in the range `1-65535` or `unlimited`. Processing continues if this limit is reached, but further warnings are not reported. The default value is `100`.

5.5 JSON/Avro Commands

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

- avroextractschema 136: Extracts the Avro schema from an Avro binary file
- json2xml 139: Converts a JSON instance document to an XML instance document.
- jsonschema2xsd 144: Converts a JSON Schema document to an XML Schema document.
- valavro 148: Validates the data in one or more Avro binaries against the respective Avro schema of each binary
- valavrojson 152: Validates one or more JSON data files against an Avro schema
- valavroschema 156: Validates an Avro schema against the Avro schema specification
- valjsonschema 159: Checks the validity of JSON schema documents
- valjson 163: Checks the validity of JSON documents
- wfjson 168: Checks the well-formedness of JSON documents.
- xml2json 172: Converts an XML instance document to a JSON instance document.
- xsd2jsonschema 176: Converts an XML Schema document to a JSON Schema document.

5.5.1 avroextractschema

An Avro binary file contains an Avro data block preceded by the Avro schema that defines the structure of the data block. The **avroextractschema** command extracts the Avro schema from the Avro binary and serializes the Avro schema as JSON.

```
raptorxml avroextractschema [options] --avrooutput=AvroSchemaFile AvroBinaryFile
```

- The *AvroBinaryFile* argument specifies the Avro binary file from which the Avro schema is to be extracted.
- The `--avrooutput` option specifies the location of the extracted Avro schema.

Example

Example of the **avroextractschema** command:

- **raptorxml** `avroextractschema --avrooutput=c:\MyAvroSchema.avsc c:\MyAvroBinary.avro`

▼ Casing and slashes on the command line

RaptorXML (and **RaptorXMLServer** for administration commands) *on Windows*
raptorxml (and **raptorxmlserver** for administration commands) *on Windows and Unix (Linux, Mac)*

* Note that lowercase (`raptorxml` and `raptorxmlserver`) works on all platforms (Windows, Linux, and Mac), while upper-lower (`RaptorXML`) works only on Windows and Mac.
* Use forward slashes on Linux and Mac, backslashes on Windows.

▼ Backslashes, spaces, and special characters on Windows systems

On Windows systems: When spaces or special characters occur in strings (for example in file or folder

© 2017-2023 Altova GmbH

names, or company, person or product names), use quotes: for example, **"My File"**. Note, however, that a backslash followed by a double-quotation mark (for example, **"C:\My directory\"**) might not be read correctly. This is because the backslash character is also used to indicate the start of an escape sequence, and the escape sequence **\"** stands for the double-quotation mark character. If you want to escape this sequence of characters, use a preceding backslash, like this: **\\"**. To summarize: If you need to write a file path that contains spaces or an end backslash, write it like this: **"C:\My Directory\ \"**.

Options

Options are listed in short form (if available) and long form. You can use one or two dashes for both short and long forms. An option may or may not take a value. If it takes a value, it is written like this: **--option=value**. 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. If an option takes a Boolean value and no value is specified, then the option's default value is `TRUE`. Use the **-h, --help** option to display information about the command.

- Processing
  - output**, avrooutput**

    --output = *FILE*, **--avrooutput =** *FILE*
    Sets the location of the Avro output file.

  - recurse

    --recurse = `true|false`
    Used to select files within sub-directories, including in ZIP archives. 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. References to ZIP files must be given in quotes. The wildcard characters `*` and `?` may be used. So, `*.xml` will select all `.xml` files in the (zip) folder. The option'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\RaptorXMLServer2024\etc\RootCatalog.xml`). See the section, XML Catalogs [47], 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 [47], for information about working with catalogs.

  - enable-globalresources

--enable-globalresources = `true|false`
Enables global resources[53]. 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[53] (and enables global resources)[53].

- globalresourcefile [gr]

  --gr | --globalresourcefile = *FILE*
  Specifies the global resource file[53] (and enables global resources)[53].

- Common options
  - 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 | unlimited*
    Specifies the error limit with a value range of `1` to `9999` or `unlimited`. The default value is `100`. Processing stops when the error limit is reached. Useful for limiting processor use during validation/transformation.

  - info-limit

    --info-limit = *N | unlimited*
    Specifies the information message limit in the range `1-65535` or `unlimited`. Processing continues if the specified info limit is reached, but further messages are not reported. The default value is `100`.

  - 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`.)

  - 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.

  - log-output

    --log-output = *FILE*

 © 2017-2023 Altova GmbH

Writes the log output to the specified file URL. Ensure that the CLI has write permission to the output location.

- network-timeout

  --network-timeout = *VALUE*
  Specifies the timeout in milliseconds for remote I/O operations. Default is: `40000`.

- recurse

  --recurse = `true|false`
  Used to select files within sub-directories, including in ZIP archives. 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. References to ZIP files must be given in quotes. The wildcard characters `*` and `?` may be used. So, `*.xml` will select all `.xml` files in the (zip) folder. The option's default value is `false`.
  Note: Boolean option values are set to `true` if the option is specified without a value.

- 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 Server. If used with a command, place `--version` before the command.

- warning-limit

  --warning-limit = *N | unlimited*
  Specifies the warning limit in the range `1-65535` or `unlimited`. Processing continues if this limit is reached, but further warnings are not reported. The default value is `100`.

5.5.2 json2xml

The **json2xml** command converts a JSON instance document to an XML document.

```
raptorxml json2xml [options] JSONFile
```

- The *JSONFile* argument is the JSON file to convert.

- Use the `--conversion-output` option to specify the location of the generated XML file.

Example

Example of the **`json2xml`** command:

- **`raptorxml`** `json2xml --conversion-output=c:\MyXMLData.xml c:\MyJSONData.json`

▼ Casing and slashes on the command line

`RaptorXML` (and **`RaptorXMLServer`** for administration commands) *on Windows*
`raptorxml` (and **`raptorxmlserver`** for administration commands) *on Windows and Unix (Linux, Mac)*

* Note that lowercase (`raptorxml` and `raptorxmlserver`) works on all platforms (Windows, Linux, and Mac), while upper-lower (`RaptorXML`) works only on Windows and Mac.
* Use forward slashes on Linux and Mac, backslashes on Windows.

▼ Backslashes, spaces, and special characters on Windows systems

On Windows systems: When spaces or special characters occur in strings (for example in file or folder names, or company, person or product names), use quotes: for example, **`"My File"`**. Note, however, that a backslash followed by a double-quotation mark (for example, **`"C:\My directory\"`**) might not be read correctly. This is because the backslash character is also used to indicate the start of an escape sequence, and the escape sequence **`\"`** stands for the double-quotation mark character. If you want to escape this sequence of characters, use a preceding backslash, like this: **`\\"`**. To summarize: If you need to write a file path that contains spaces or an end backslash, write it like this: **`"C:\My Directory\ \"`**.

Options

Options are listed in short form (if available) and long form. You can use one or two dashes for both short and long forms. An option may or may not take a value. If it takes a value, it is written like this: **`--option=value`**. 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. If an option takes a Boolean value and no value is specified, then the option's default value is `TRUE`. Use the **`--h, --help`** option to display information about the command.

▼ JSON to XML conversion options

These options define the handling of specific conversion-related details in conversions between XML and JSON.

▼ array-element

`--array-element =` `VALUE`
Specifies the name of the element to be converted to an array item.

▼ attributes

`--attributes =` `true|false`
If set to **`true`**, then conversion between XML attributes and JSON @-prefixed properties occurs. Otherwise, XML attributes and JSON @-properties will not be converted. The default is **`true`**.
Note: Boolean option values are set to `true` if the option is specified without a value.

 © 2017-2023 Altova GmbH

- comments

  --comments = true|false
  If set to **true**, then conversion between XML comments and JSON #-prefixed properties occurs. Otherwise, XML attributes and JSON #-properties will not be converted. The default is **true**.
  Note: Boolean option values are set to true if the option is specified without a value.

- conversion-output, o

  --o, --converson-output = FILE
  Sets the path and name of the file to which the result of the conversion is sent.

- create-array-container

  --create-array-container = true|false
  If set to **true**, creates a container element in the generated XML file for every JSON array in the source JSON document. The default is **false**.
  Note: Boolean option values are set to true if the option is specified without a value.

- encode-colons

  --encode-colons = true|false
  If set to **true**, colons in JSON property names are encoded in the generated XML document. The default is **true**.
  Note: Boolean option values are set to true if the option is specified without a value.

- json-type-hints

  --json-type-hints = true|false
  If set to **true**, adds attributes in the generated XML document for type-hints in the source JSON document. The default is **true**.
  Note: Boolean option values are set to true if the option is specified without a value.

- pi

  --pi = true|false
  If set to **true**, then conversion between XML processing instructions and JSON ?-prefixed properties occurs. Otherwise, XML attributes and JSON ?-properties will not be converted. The default is **true**.
  Note: Boolean option values are set to true if the option is specified without a value.

- pretty-print

  --pp, --pretty-print = true|false
  If set to **true**, pretty-prints the generated output document. The default is **false**.
  Note: Boolean option values are set to true if the option is specified without a value.

- text

  --text = true|false
  If set to **true**, then conversion between XML text content and JSON $-prefixed properties occurs. Otherwise, XML attributes and JSON $-properties will not be converted. The default is **true**.
  Note: Boolean option values are set to true if the option is specified without a value.

- Common options
  - 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 | unlimited*
    Specifies the error limit with a value range of 1 to 9999 or unlimited. The default value is 100. Processing stops when the error limit is reached. Useful for limiting processor use during validation/transformation.

  - info-limit

    --info-limit = *N | unlimited*
    Specifies the information message limit in the range 1-65535 or unlimited. Processing continues if the specified info limit is reached, but further messages are not reported. The default value is 100.

  - 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.)

  - 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.

  - log-output

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

  - network-timeout

    --network-timeout = *VALUE*
    Specifies the timeout in milliseconds for remote I/O operations. Default is: 40000.

  - recurse

    --recurse = true|false
    Used to select files within sub-directories, including in ZIP archives. If true, the command's *InputFile* argument will select the specified file also in subdirectories. For example: "test.zip|

 © 2017-2023 Altova GmbH

zip\test.xml" will select files named test.xml at all folder levels of the zip folder. References to ZIP files must be given in quotes. The wildcard characters * and ? may be used. So, *.xml will select all .xml files in the (zip) folder. The option's default value is false.
Note: Boolean option values are set to true if the option is specified without a value.

- 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 Server. If used with a command, place --version before the command.

- warning-limit

  --warning-limit = *N | unlimited*
  Specifies the warning limit in the range 1-65535 or unlimited. Processing continues if this limit is reached, but further warnings are not reported. The default value is 100.

- 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\RaptorXMLServer2024\etc\RootCatalog.xml). See the section, XML Catalogs [47], 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 [47], for information about working with catalogs.

  - enable-globalresources

    --enable-globalresources = true|false
    Enables global resources [53]. 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[53] (and enables global resources)[53].

▼ globalresourcefile [gr]

--gr | --globalresourcefile = *FILE*
Specifies the global resource file[53] (and enables global resources)[53].

5.5.3 jsonschema2xsd

The **jsonschema2xsd** command converts a JSON Schema document to an XML Schema document that conforms with the rules of the W3C XSD 1.0 and 1.1 specifications.

```
raptorxml jsonschema2xsd [options] JSONSchemaFile
```

- The *JSONSchemaFile* argument is the JSON Schema file to convert.
- Use the `--schema-conversion-output` option to specify the location of the generated XSD file.

Example

Example of the **jsonschema2xsd** command:

- **raptorxml** jsonschema2xsd --schema-conversion-output=c:\MyXMLSchema.xsd c:\MyJSONSchema.json

▼ Casing and slashes on the command line

RaptorXML (and **RaptorXMLServer** for administration commands) *on Windows*
raptorxml (and **raptorxmlserver** for administration commands) *on Windows and Unix (Linux, Mac)*

* Note that lowercase (raptorxml and raptorxmlserver) works on all platforms (Windows, Linux, and Mac), while upper-lower (RaptorXML) works only on Windows and Mac.
* Use forward slashes on Linux and Mac, backslashes on Windows.

▼ Backslashes, spaces, and special characters on Windows systems

On Windows systems: When spaces or special characters occur in strings (for example in file or folder names, or company, person or product names), use quotes: for example, **"My File"**. Note, however, that a backslash followed by a double-quotation mark (for example, **"C:\My directory\"**) might not be read correctly. This is because the backslash character is also used to indicate the start of an escape sequence, and the escape sequence **\"** stands for the double-quotation mark character. If you want to escape this sequence of characters, use a preceding backslash, like this: **\\"**. To summarize: If you need to write a file path that contains spaces or an end backslash, write it like this: **"C:\My Directory\\"**.

 © 2017-2023 Altova GmbH

Options

Options are listed in short form (if available) and long form. You can use one or two dashes for both short and long forms. An option may or may not take a value. If it takes a value, it is written like this: **--option=value**. 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. If an option takes a Boolean value and no value is specified, then the option's default value is `TRUE`. Use the **--h, --help** option to display information about the command.

JSON validation options

These are options for validating the source JSON Schema document.

additional-schema

--additional-schema = *FILE*

Specifies URIs of an additional schema document. The additional schema will be loaded by the main schema and can be referenced from the main schema by the additional schemas **id** or **$id** property.

disable-format-checks

--disable-format-checks = `true|false`

Disables the semantic validation imposed by the format attribute. Default value is `false`.
Note: Boolean option values are set to `true` if the option is specified without a value.

jsonschema-version

--jsonschema-version = `draft04|draft06|draft07|2019-09|2020-12|latest|detect`

Specifies which version of the JSON Schema specification draft version to use. Default is `detect`.

strict-integer-checks

--strict-integer-checks = `true|false`

Specifies whether the stricter integer checks of draft-04 should be used with later schemas—where integer checks are looser. For example, `1.0` is not a valid integer in draft-04, but is a valid integer in later drafts. This option has no effect for draft-04 schemas. The default value of the option is `false`.
Note: Boolean option values are set to `true` if the option is specified without a value.

Conversion from JSON Schema to XSD

These are options to specify details of the JSON Schema to XSD conversion.

at-to-attributes

--at-to-attributes = `true|false`

If set to **true**, then properties prefixed with @ in the JSON Schema document are converted to attributes in the generated XSD document. The default is **true**.
Note: Boolean option values are set to `true` if the option is specified without a value.

dollar-to-text

--dollar-to-text = `true|false`

If set to **true**, then $-prefixed properties in the JSON Schema document are converted to text in the generated XSD document. The default is **true**.

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

- ignore-comments

  `--ignore-comments =` `true|false`
  If set to **`true`**, ignores properties in the source JSON Schema named `'#'`. The default is **`true`**.
  Note: Boolean option values are set to `true` if the option is specified without a value.

- ignore-pi-proprties

  `--ignore-pi-properties =` `true|false`
  If set to **`true`**, ignores properties in the source JSON Schema document that start with `'?'`. The default is **`true`**.
  Note: Boolean option values are set to `true` if the option is specified without a value.

- ignore-xmlns-proprties

  `--ignore-xmlns-properties =` `true|false`
  If set to **`true`**, ignores properties in the source JSON Schema document that start with `'@xmlns'`. The default is **`true`**.
  Note: Boolean option values are set to `true` if the option is specified without a value.

- ignore-xsi-proprties

  `--ignore-xsi-properties =` `true|false`
  If set to **`true`**, ignores properties in the source JSON Schema document that start with `'@xsi'`. The default is **`true`**.
  Note: Boolean option values are set to `true` if the option is specified without a value.

- schema-conversion-output, o

  `--o, --schema-converson-output =` `FILE`
  Sets the path and name of the file to which the result of the conversion is sent.

- Common options

  - 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 | unlimited`*
    Specifies the error limit with a value range of `1` to `9999` or `unlimited`. The default value is `100`. Processing stops when the error limit is reached. Useful for limiting processor use during validation/transformation.

  - info-limit

    `--info-limit =` *`N | unlimited`*
    Specifies the information message limit in the range `1-65535` or `unlimited`. Processing continues if

 © 2017-2023 Altova GmbH

the specified info limit is reached, but further messages are not reported. The default value is `100`.

- 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`.)

- 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.

- log-output

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

- network-timeout

  `--network-timeout =` *`VALUE`*
  Specifies the timeout in milliseconds for remote I/O operations. Default is: `40000`.

- recurse

  `--recurse =` `true|false`
  Used to select files within sub-directories, including in ZIP archives. 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. References to ZIP files must be given in quotes. The wildcard characters `*` and `?` may be used. So, `*.xml` will select all `.xml` files in the (zip) folder. The option's default value is `false`.
  Note: Boolean option values are set to `true` if the option is specified without a value.

- 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 Server. If used with a command, place `--version` before the command.

- warning-limit

  --warning-limit = *N | unlimited*
  Specifies the warning limit in the range `1-65535` or `unlimited`. Processing continues if this limit is reached, but further warnings are not reported. The default value is `100`.

- 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\RaptorXMLServer2024\etc\RootCatalog.xml`). See the section, XML Catalogs [47], 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 [47], for information about working with catalogs.

  - enable-globalresources

    --enable-globalresources = `true|false`
    Enables global resources [53]. 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 [53] (and enables global resources) [53].

  - globalresourcefile [gr]

    --gr | --globalresourcefile = *FILE*
    Specifies the global resource file [53] (and enables global resources) [53].

5.5.4 valavro (avro)

The **valavro | avro** command validates the data block in one or more Avro binary files against the respective Avro schemas in each binary file.

```
raptorxml valavro | avro [options] AvroBinaryFile
```


© 2017-2023 Altova GmbH

- The *AvroBinaryFile* argument specifies one or more Avro binary files to validate. Specifically, the data block in each Avro binary file is validated against the Avro schema in that binary file.
- To validate multiple Avro binaries, 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 *AvroBinaryFile* argument together with the `--listfile`[223] option set to `true` (*see the Options list below*).

Examples

Examples of the **valavro** command:

- **raptorxml** valavro c:\MyAvroBinary.avro
- **raptorxml** valavro c:\MyAvroBinary01.avro c:\MyAvroBinary02.avro
- **raptorxml** avro --listfile=true c:\MyFileList.txt

▼ Casing and slashes on the command line

RaptorXML (and **RaptorXMLServer** for administration commands) *on Windows*
raptorxml (and **raptorxmlserver** for administration commands) *on Windows and Unix (Linux, Mac)*

* Note that lowercase (`raptorxml` and `raptorxmlserver`) works on all platforms (Windows, Linux, and Mac), while upper-lower (`RaptorXML`) works only on Windows and Mac.
* Use forward slashes on Linux and Mac, backslashes on Windows.

▼ Backslashes, spaces, and special characters on Windows systems

On Windows systems: When spaces or special characters occur in strings (for example in file or folder names, or company, person or product names), use quotes: for example, **"My File"**. Note, however, that a backslash followed by a double-quotation mark (for example, **"C:\My directory\"**) might not be read correctly. This is because the backslash character is also used to indicate the start of an escape sequence, and the escape sequence **\"** stands for the double-quotation mark character. If you want to escape this sequence of characters, use a preceding backslash, like this: **\\"**. To summarize: If you need to write a file path that contains spaces or an end backslash, write it like this: **"C:\My Directory\\"**.

Options

Options are listed in short form (if available) and long form. You can use one or two dashes for both short and long forms. An option may or may not take a value. If it takes a value, it is written like this: **--option=value**. 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. If an option takes a Boolean value and no value is specified, then the option's default value is `TRUE`. Use the **--h, --help** option to display information about the command.

▼ 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 sub-directories, including in ZIP archives. 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. References to ZIP files must be given in quotes. The wildcard characters `*` and `?` may be used. So, `*.xml` will select all `.xml` files in the (zip) folder. The option'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\RaptorXMLServer2024\etc\RootCatalog.xml`). See the section, XML Catalogs [47], 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 [47], for information about working with catalogs.

  - enable-globalresources

    `--enable-globalresources =` `true|false`
    Enables global resources [53]. 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 [53] (and enables global resources) [53].

  - globalresourcefile [gr]

    `--gr | --globalresourcefile =` *`FILE`*
    Specifies the global resource file [53] (and enables global resources) [53].

- Common options
  - 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.

© 2017-2023 Altova GmbH

- error-limit

  --error-limit = *N | unlimited*
  Specifies the error limit with a value range of `1` to `9999` or `unlimited`. The default value is `100`. Processing stops when the error limit is reached. Useful for limiting processor use during validation/transformation.

- info-limit

  --info-limit = *N | unlimited*
  Specifies the information message limit in the range `1-65535` or `unlimited`. Processing continues if the specified info limit is reached, but further messages are not reported. The default value is `100`.

- 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`.)

- 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.

- log-output

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

- network-timeout

  --network-timeout = *VALUE*
  Specifies the timeout in milliseconds for remote I/O operations. Default is: `40000`.

- recurse

  --recurse = `true|false`
  Used to select files within sub-directories, including in ZIP archives. 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. References to ZIP files must be given in quotes. The wildcard characters `*` and `?` may be used. So, `*.xml` will select all `.xml` files in the (zip) folder. The option's default value is `false`.
  Note: Boolean option values are set to `true` if the option is specified without a value.

- 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 Server. If used with a command, place `--version` before the command.

- warning-limit

  `--warning-limit =` *`N | unlimited`*
  Specifies the warning limit in the range `1-65535` or `unlimited`. Processing continues if this limit is reached, but further warnings are not reported. The default value is `100`.

5.5.5 valavrojson (avrojson)

The **`valavrojson | avrojson`** command validates a JSON document against an Avro schema.

```
raptorxml valavrojson | avrojson [options] --avroschema=AvroSchema JSONFile
```

- The *`JSONFile`* argument specifies the JSON document to validate.
- The **`--avroschema`** option specifies the Avro schema against which the JSON document is to be validated.
- To validate multiple JSON files, either: (i) list the files on the CLI, with each file separated from the next by a space; or (ii) list the files to validate in a text file (`.txt` file), with one filename per line, and supply this text file as the *`JSONFile`* argument together with the `--listfile` 223 option set to `true` (*see the Options list below*).

Examples

Examples of the **`valavrojson`** command:

- **`raptorxml`** `valavrojson --avroschema=c:\MyAvroSchema.avsc c:\MyJSONDataFile.json`
- **`raptorxml`** `avrojson --avroschema=c:\MyAvroSchema.avsc c:\MyJSONDataFile.json`

- Casing and slashes on the command line

  `RaptorXML` (and **`RaptorXMLServer`** for administration commands) *on Windows*
  `raptorxml` (and **`raptorxmlserver`** for administration commands) *on Windows and Unix (Linux, Mac)*

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

 © 2017-2023 Altova GmbH

* Use forward slashes on Linux and Mac, backslashes on Windows.

▼ Backslashes, spaces, and special characters on Windows systems

On Windows systems: When spaces or special characters occur in strings (for example in file or folder names, or company, person or product names), use quotes: for example, **"My File"**. Note, however, that a backslash followed by a double-quotation mark (for example, **"C:\My directory\"**) might not be read correctly. This is because the backslash character is also used to indicate the start of an escape sequence, and the escape sequence **\"** stands for the double-quotation mark character. If you want to escape this sequence of characters, use a preceding backslash, like this: **\\"**. To summarize: If you need to write a file path that contains spaces or an end backslash, write it like this: **"C:\My Directory\ \"**.

Options

Options are listed in short form (if available) and long form. You can use one or two dashes for both short and long forms. An option may or may not take a value. If it takes a value, it is written like this: **--option=value**. 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. If an option takes a Boolean value and no value is specified, then the option's default value is `TRUE`. Use the **--h, --help** option to display information about the command.

▼ 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 sub-directories, including in ZIP archives. 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. References to ZIP files must be given in quotes. The wildcard characters `*` and `?` may be used. So, `*.xml` will select all `.xml` files in the (zip) folder. The option'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\RaptorXMLServer2024\etc\RootCatalog.xml`). See the section, XML

Catalogs[47], 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[47], for information about working with catalogs.

- enable-globalresources

  --enable-globalresources = `true|false`
  Enables global resources[53]. 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[53] (and enables global resources)[53].

- globalresourcefile [gr]

  --gr | --globalresourcefile = *FILE*
  Specifies the global resource file[53] (and enables global resources)[53].

- Common options

  - 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 | unlimited*
    Specifies the error limit with a value range of `1` to `9999` or `unlimited`. The default value is `100`. Processing stops when the error limit is reached. Useful for limiting processor use during validation/transformation.

  - info-limit

    --info-limit = *N | unlimited*
    Specifies the information message limit in the range `1-65535` or `unlimited`. Processing continues if the specified info limit is reached, but further messages are not reported. The default value is `100`.

  - 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`.)

  - listfile

    --listfile = `true|false`

 © 2017-2023 Altova GmbH

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.

- log-output

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

- network-timeout

  `--network-timeout =` *`VALUE`*
  Specifies the timeout in milliseconds for remote I/O operations. Default is: `40000`.

- recurse

  `--recurse =` `true|false`
  Used to select files within sub-directories, including in ZIP archives. 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. References to ZIP files must be given in quotes. The wildcard characters `*` and `?` may be used. So, `*.xml` will select all `.xml` files in the (zip) folder. The option's default value is `false`.
  Note: Boolean option values are set to `true` if the option is specified without a value.

- 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 Server. If used with a command, place `--version` before the command.

- warning-limit

  `--warning-limit =` *`N | unlimited`*
  Specifies the warning limit in the range `1-65535` or `unlimited`. Processing continues if this limit is reached, but further warnings are not reported. The default value is `100`.

5.5.6 valavroschema (avroschema)

The **valavroschema | avroschema** command validates one or more Avro schema documents against the Avro schema specification.

```
raptorxml valavroschema | avroschema [options] AvroSchema
```

- The *AvroSchema* argument is the Avro schema document to validate.
- To validate multiple Avro schemas, 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 *AvroSchema* argument together with the `--listfile`[223] option set to `true` (*see the Options list below*).

Examples

Examples of the **valavroschema** command:

- **raptorxml** valavroschema c:\MyAvroSchema.avsc
- **raptorxml** valavroschema c:\MyAvroSchema01.avsc c:\MyAvroSchema02.avsc
- **raptorxml** avroschema --listfile=true c:\MyFileList.txt

▼ Casing and slashes on the command line

RaptorXML (and **RaptorXMLServer** for administration commands) *on Windows*
raptorxml (and **raptorxmlserver** for administration commands) *on Windows and Unix (Linux, Mac)*

* Note that lowercase (`raptorxml` and `raptorxmlserver`) works on all platforms (Windows, Linux, and Mac), while upper-lower (`RaptorXML`) works only on Windows and Mac.
* Use forward slashes on Linux and Mac, backslashes on Windows.

▼ Backslashes, spaces, and special characters on Windows systems

On Windows systems: When spaces or special characters occur in strings (for example in file or folder names, or company, person or product names), use quotes: for example, **"My File"**. Note, however, that a backslash followed by a double-quotation mark (for example, **"C:\My directory\"**) might not be read correctly. This is because the backslash character is also used to indicate the start of an escape sequence, and the escape sequence **\"** stands for the double-quotation mark character. If you want to escape this sequence of characters, use a preceding backslash, like this: **\\"**. To summarize: If you need to write a file path that contains spaces or an end backslash, write it like this: **"C:\My Directory\ \"**.

Options

Options are listed in short form (if available) and long form. You can use one or two dashes for both short and long forms. An option may or may not take a value. If it takes a value, it is written like this: **--option=value**. 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. If an option takes a Boolean value and no value is specified, then the option's default value is `TRUE`. Use the **--h, --help** option to display information about the command.

© 2017-2023 Altova GmbH

- 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 sub-directories, including in ZIP archives. 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. References to ZIP files must be given in quotes. The wildcard characters `*` and `?` may be used. So, `*.xml` will select all `.xml` files in the (zip) folder. The option'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\RaptorXMLServer2024\etc\RootCatalog.xml`). See the section, XML Catalogs 47, 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 47, for information about working with catalogs.

  - enable-globalresources

    --enable-globalresources = `true|false`
    Enables global resources 53. 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 53 (and enables global resources) 53.

  - globalresourcefile [gr]

    --gr | --globalresourcefile = *FILE*
    Specifies the global resource file 53 (and enables global resources) 53.

- Common options
  - 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 | unlimited*
    Specifies the error limit with a value range of 1 to 9999 or unlimited. The default value is 100. Processing stops when the error limit is reached. Useful for limiting processor use during validation/transformation.

  - info-limit

    --info-limit = *N | unlimited*
    Specifies the information message limit in the range 1-65535 or unlimited. Processing continues if the specified info limit is reached, but further messages are not reported. The default value is 100.

  - 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.)

  - 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.

  - log-output

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

  - network-timeout

    --network-timeout = *VALUE*
    Specifies the timeout in milliseconds for remote I/O operations. Default is: 40000.

  - recurse

    --recurse = true|false
    Used to select files within sub-directories, including in ZIP archives. 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. References to

 © 2017-2023 Altova GmbH

ZIP files must be given in quotes. The wildcard characters `*` and `?` may be used. So, `*.xml` will select all `.xml` files in the (zip) folder. The option's default value is `false`.
Note: Boolean option values are set to `true` if the option is specified without a value.

- 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 Server. If used with a command, place `--version` before the command.

- warning-limit

  `--warning-limit =` *`N | unlimited`*
  Specifies the warning limit in the range `1-65535` or `unlimited`. Processing continues if this limit is reached, but further warnings are not reported. The default value is `100`.

5.5.7 valjsonschema (jsonschema)

The **`valjsonschema | jsonschema`** command validates one or more JSON schema documents according to the various JSON Schema specifications (set via the **`jsonschema-version`** option.

```
raptorxml valjsonschema | jsonschema [options] InputFile
```

- The *`InputFile`* argument is the JSON schema 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`[223] option set to `true` (*see the Options list below*).

Examples

Examples of the **`valjsonschema`** command:

- **`raptorxml`** `valjsonschema c:\MyJSONSchema.json`
- **`raptorxml`** `jsonschema c:\MyJSONSchema-01.json c:\MyJSONSchema-02.json`
- **`raptorxml`** `jsonschema --listfile=true c:\FileList.txt`

- Casing and slashes on the command line

  RaptorXML (and **RaptorXMLServer** for administration commands) *on Windows*
  raptorxml (and **raptorxmlserver** for administration commands) *on Windows and Unix (Linux, Mac)*

  * Note that lowercase (`raptorxml` and `raptorxmlserver`) works on all platforms (Windows, Linux, and Mac), while upper-lower (`RaptorXML`) works only on Windows and Mac.
  * Use forward slashes on Linux and Mac, backslashes on Windows.

- Backslashes, spaces, and special characters on Windows systems

  On Windows systems: When spaces or special characters occur in strings (for example in file or folder names, or company, person or product names), use quotes: for example, **"My File"**. Note, however, that a backslash followed by a double-quotation mark (for example, **"C:\My directory\"**) might not be read correctly. This is because the backslash character is also used to indicate the start of an escape sequence, and the escape sequence **\"** stands for the double-quotation mark character. If you want to escape this sequence of characters, use a preceding backslash, like this: **\\"**. To summarize: If you need to write a file path that contains spaces or an end backslash, write it like this: **"C:\My Directory\ \"**.

Options

Options are listed in short form (if available) and long form. You can use one or two dashes for both short and long forms. An option may or may not take a value. If it takes a value, it is written like this: **--option=value**. 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. If an option takes a Boolean value and no value is specified, then the option's default value is `TRUE`. Use the **--h, --help** option to display information about the command.

- 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 sub-directories, including in ZIP archives. 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. References to ZIP files must be given in quotes. The wildcard characters `*` and `?` may be used. So, `*.xml` will select all `.xml` files in the (zip) folder. The option's default value is `false`.
    Note: Boolean option values are set to `true` if the option is specified without a value.

© 2017-2023 Altova GmbH

- JSON validation options
  - additional-schema

    `--additional-schema =` *FILE*
    Specifies URIs of an additional schema document. The additional schema will be loaded by the main schema and can be referenced from the main schema by the additional schemas **`id`** or **`$id`** property.

  - disable-format-checks

    `--disable-format-checks =` `true|false`
    Disables the semantic validation imposed by the format attribute. Default value is `false`.
    Note: Boolean option values are set to `true` if the option is specified without a value.

  - jsonschema-version

    `--jsonschema-version =` `draft04|draft06|draft07|2019-09|2020-12|latest|detect`
    Specifies which version of the JSON Schema specification draft version to use. Default is `detect`.

  - strict-integer-checks

    `--strict-integer-checks =` `true|false`
    Specifies whether the stricter integer checks of draft-04 should be used with later schemas—where integer checks are looser. For example, `1.0` is not a valid integer in draft-04, but is a valid integer in later drafts. This option has no effect for draft-04 schemas. The default value of the option 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\RaptorXMLServer2024\etc\RootCatalog.xml`). See the section, XML Catalogs 47, 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 47, for information about working with catalogs.

  - enable-globalresources

    `--enable-globalresources =` `true|false`
    Enables global resources 53. 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 53 (and enables global resources) 53.

- globalresourcefile [gr]

  --gr | --globalresourcefile = *FILE*
  Specifies the global resource file[53] (and enables global resources)[53].

- Common options
  - 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 | unlimited*
    Specifies the error limit with a value range of `1` to `9999` or `unlimited`. The default value is `100`. Processing stops when the error limit is reached. Useful for limiting processor use during validation/transformation.

  - info-limit

    --info-limit = *N | unlimited*
    Specifies the information message limit in the range `1-65535` or `unlimited`. Processing continues if the specified info limit is reached, but further messages are not reported. The default value is `100`.

  - 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`.)

  - 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.

  - log-output

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

  - network-timeout

    --network-timeout = *VALUE*
    Specifies the timeout in milliseconds for remote I/O operations. Default is: `40000`.

  - recurse

© 2017-2023 Altova GmbH

--recurse = true|false
Used to select files within sub-directories, including in ZIP archives. 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. References to ZIP files must be given in quotes. The wildcard characters * and ? may be used. So, *.xml will select all .xml files in the (zip) folder. The option's default value is false.
Note: Boolean option values are set to true if the option is specified without a value.

- 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 Server. If used with a command, place --version before the command.

- warning-limit

  --warning-limit = *N | unlimited*
  Specifies the warning limit in the range 1-65535 or unlimited. Processing continues if this limit is reached, but further warnings are not reported. The default value is 100.

5.5.8 valjson (json)

The **valjson | json** command validates one or more JSON instance documents according to the JSON schema supplied with the --schema option.

```
raptorxml valjson | json [options] --jsonschema=File InputFile
```

- The *InputFile* argument is the JSON 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[223] option set to true (*see the Options list below*).

Examples

Examples of the **valjson** command:

- **raptorxml** valjson --jsonschema=c:\MyJSONSchema.json c:\MyJSONInstance.json
- **raptorxml** json --jsonschema=c:\MyJSONSchema.json c:\MyJSONInstance-01.json c:\MyJSONInstance-02.json
- **raptorxml** json --jsonschema=c:\MyJSONSchema.json --listfile=true c:\FileList.txt

▼ Casing and slashes on the command line

RaptorXML (and **RaptorXMLServer** for administration commands) *on Windows*
raptorxml (and **raptorxmlserver** for administration commands) *on Windows and Unix (Linux, Mac)*

* Note that lowercase (raptorxml and raptorxmlserver) works on all platforms (Windows, Linux, and Mac), while upper-lower (RaptorXML) works only on Windows and Mac.
* Use forward slashes on Linux and Mac, backslashes on Windows.

▼ Backslashes, spaces, and special characters on Windows systems

On Windows systems: When spaces or special characters occur in strings (for example in file or folder names, or company, person or product names), use quotes: for example, **"My File"**. Note, however, that a backslash followed by a double-quotation mark (for example, **"C:\My directory\"**) might not be read correctly. This is because the backslash character is also used to indicate the start of an escape sequence, and the escape sequence **\"** stands for the double-quotation mark character. If you want to escape this sequence of characters, use a preceding backslash, like this: **\\"**. To summarize: If you need to write a file path that contains spaces or an end backslash, write it like this: **"C:\My Directory\\"**.

Options

Options are listed in short form (if available) and long form. You can use one or two dashes for both short and long forms. An option may or may not take a value. If it takes a value, it is written like this: **--option=value**. 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. If an option takes a Boolean value and no value is specified, then the option's default value is TRUE. Use the **--h, --help** option to display information about the command.

▼ Validation and processing

▼ schema, **jsonschema**

--schema = *FILE*, **--jsonschema =** *FILE*
Specifies the path to the JSON Schema document to use for the validation of JSON instance documents.

▼ 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.

 © 2017-2023 Altova GmbH

- additional-schema

  --additional-schema = *FILE*
  Specifies URIs of an additional schema document. The additional schema will be loaded by the main schema and can be referenced from the main schema by the additional schemas **id** or **$id** property.

- recurse

  --recurse = true|false
  Used to select files within sub-directories, including in ZIP archives. 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. References to ZIP files must be given in quotes. The wildcard characters * and ? may be used. So, *.xml will select all .xml files in the (zip) folder. The option's default value is false.
  Note: Boolean option values are set to true if the option is specified without a value.

- json5

  --json5 = true|false
  Enables JSON5 support. Default value is false.
  Note: Boolean option values are set to true if the option is specified without a value.

- jsonc

  --jsonc = true|false
  Enables support for comments in JSON. Default value is false.
  Note: Boolean option values are set to true if the option is specified without a value.

- json-lines

  --json-lines = true|false
  Enables support for JSON Lines (that is, one JSON value per line). Default value is false.
  Note: Boolean option values are set to true if the option is specified without a value.

- jsonschema-version

  --jsonschema-version = draft04|draft06|draft07|2019-09|2020-12|latest|detect
  Specifies which version of the JSON Schema specification draft version to use. Default is detect.

- disable-format-checks

  --disable-format-checks = true|false
  Disables the semantic validation imposed by the format attribute. Default value is false.
  Note: Boolean option values are set to true if the option is specified without a value.

- strict-integer-checks

  --strict-integer-checks = true|false
  Specifies whether the stricter integer checks of draft-04 should be used with later schemas—where integer checks are looser. For example, **1.0** is not a valid integer in draft-04, but is a valid integer in later drafts. This option has no effect for draft-04 schemas. The default value of the option 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\RaptorXMLServer2024\etc\RootCatalog.xml`). See the section, XML Catalogs [47], 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 [47], for information about working with catalogs.

  - enable-globalresources

    --enable-globalresources = `true|false`
    Enables global resources [53]. 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 [53] (and enables global resources) [53].

  - globalresourcefile [gr]

    --gr | --globalresourcefile = *FILE*
    Specifies the global resource file [53] (and enables global resources) [53].

- Common options
  - 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 | unlimited*
    Specifies the error limit with a value range of `1` to `9999` or `unlimited`. The default value is `100`. Processing stops when the error limit is reached. Useful for limiting processor use during validation/transformation.

  - info-limit

    --info-limit = *N | unlimited*
    Specifies the information message limit in the range `1-65535` or `unlimited`. Processing continues if the specified info limit is reached, but further messages are not reported. The default value is `100`.

  - help

 © 2017-2023 Altova GmbH

--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`.)

- 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.

- log-output

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

- network-timeout

  --network-timeout = *VALUE*
  Specifies the timeout in milliseconds for remote I/O operations. Default is: `40000`.

- recurse

  --recurse = `true|false`
  Used to select files within sub-directories, including in ZIP archives. 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. References to ZIP files must be given in quotes. The wildcard characters `*` and `?` may be used. So, `*.xml` will select all `.xml` files in the (zip) folder. The option's default value is `false`.
  Note: Boolean option values are set to `true` if the option is specified without a value.

- 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 Server. If used with a command, place `--version` before the command.

- warning-limit

  --warning-limit = *N | unlimited*

Specifies the warning limit in the range `1-65535` or `unlimited`. Processing continues if this limit is reached, but further warnings are not reported. The default value is `100`.

5.5.9 wfjson

The **`wfjson`** command checks one or more JSON documents according to the ECMA-404 specification for well-formedness.

```
raptorxml wfjson [options] InputFile
```

- The *`InputFile`* argument is the JSON document (schema or instance) to check for well-formedness.
- 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` 223 option set to `true` (*see the Options list below*).

Examples

Examples of the **`wfjson`** command:

- **raptorxml** wfjson c:\MyJSONFile.json
- **raptorxml** wfjson c:\MyJSONFile-01.json c:\MyJSONFile-02.json
- **raptorxml** wfjson --listfile=true c:\FileList.txt

▼ Casing and slashes on the command line

RaptorXML (and **RaptorXMLServer** for administration commands) *on Windows*
raptorxml (and **raptorxmlserver** for administration commands) *on Windows and Unix (Linux, Mac)*

* Note that lowercase (`raptorxml` and `raptorxmlserver`) works on all platforms (Windows, Linux, and Mac), while upper-lower (`RaptorXML`) works only on Windows and Mac.
* Use forward slashes on Linux and Mac, backslashes on Windows.

▼ Backslashes, spaces, and special characters on Windows systems

On Windows systems: When spaces or special characters occur in strings (for example in file or folder names, or company, person or product names), use quotes: for example, **`"My File"`**. Note, however, that a backslash followed by a double-quotation mark (for example, **`"C:\My directory\"`**) might not be read correctly. This is because the backslash character is also used to indicate the start of an escape sequence, and the escape sequence **`\"`** stands for the double-quotation mark character. If you want to escape this sequence of characters, use a preceding backslash, like this: **`\\"`**. To summarize: If you need to write a file path that contains spaces or an end backslash, write it like this: **`"C:\My Directory\\"`**.

 © 2017-2023 Altova GmbH

Options

Options are listed in short form (if available) and long form. You can use one or two dashes for both short and long forms. An option may or may not take a value. If it takes a value, it is written like this: **`--option=value`**. 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. If an option takes a Boolean value and no value is specified, then the option's default value is `TRUE`. Use the **`--h, --help`** option to display information about the command.

- Validation and processing
  - json5

    `--json5 =` `true|false`

    Enables JSON5 support. Default value is `false`.

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

  - jsonc

    `--jsonc =` `true|false`

    Enables support for comments in JSON. Default value is `false`.

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

  - json-lines

    `--json-lines =` `true|false`

    Enables support for JSON Lines (that is, one JSON value per line). Default value is `false`.

    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.

  - recurse

    `--recurse =` `true|false`

    Used to select files within sub-directories, including in ZIP archives. 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. References to ZIP files must be given in quotes. The wildcard characters `*` and `?` may be used. So, `*.xml` will select all `.xml` files in the (zip) folder. The option'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\RaptorXMLServer2024\etc\RootCatalog.xml`). See the section, XML Catalogs[47], 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[47], for information about working with catalogs.

- enable-globalresources

  `--enable-globalresources =` `true|false`
  Enables global resources[53]. 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[53] (and enables global resources)[53].

- globalresourcefile [gr]

  `--gr | --globalresourcefile =` *`FILE`*
  Specifies the global resource file[53] (and enables global resources)[53].

- Common options
  - 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 | unlimited`*
    Specifies the error limit with a value range of `1` to `9999` or `unlimited`. The default value is `100`. Processing stops when the error limit is reached. Useful for limiting processor use during validation/transformation.

  - info-limit

    `--info-limit =` *`N | unlimited`*
    Specifies the information message limit in the range `1-65535` or `unlimited`. Processing continues if the specified info limit is reached, but further messages are not reported. The default value is `100`.

  - 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`.)

© 2017-2023 Altova GmbH

- 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.

- log-output

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

- network-timeout

  --network-timeout = *VALUE*
  Specifies the timeout in milliseconds for remote I/O operations. Default is: `40000`.

- recurse

  --recurse = `true|false`
  Used to select files within sub-directories, including in ZIP archives. 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. References to ZIP files must be given in quotes. The wildcard characters `*` and `?` may be used. So, `*.xml` will select all `.xml` files in the (zip) folder. The option's default value is `false`.
  Note: Boolean option values are set to `true` if the option is specified without a value.

- 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 Server. If used with a command, place `--version` before the command.

- warning-limit

  --warning-limit = *N | unlimited*
  Specifies the warning limit in the range `1-65535` or `unlimited`. Processing continues if this limit is reached, but further warnings are not reported. The default value is `100`.

5.5.10 xml2json

The **xml2json** command converts an XML instance document to a JSON document.

```
raptorxml XML2json [options] XMLFile
```

- The *XMLFile* argument is the XML file to convert.
- Use the `--conversion-output` option to specify the location of the generated JSON file.

Example

Example of the **xml2json** command:

- **raptorxml** xml2json --conversion-output=c:\MyJSONData.json c:\MyXMLData.xml

▼ Casing and slashes on the command line

RaptorXML (and **RaptorXMLServer** for administration commands) *on Windows*
raptorxml (and **raptorxmlserver** for administration commands) *on Windows and Unix (Linux, Mac)*

* Note that lowercase (`raptorxml` and `raptorxmlserver`) works on all platforms (Windows, Linux, and Mac), while upper-lower (`RaptorXML`) works only on Windows and Mac.
* Use forward slashes on Linux and Mac, backslashes on Windows.

▼ Backslashes, spaces, and special characters on Windows systems

On Windows systems: When spaces or special characters occur in strings (for example in file or folder names, or company, person or product names), use quotes: for example, **"My File"**. Note, however, that a backslash followed by a double-quotation mark (for example, **"C:\My directory\"**) might not be read correctly. This is because the backslash character is also used to indicate the start of an escape sequence, and the escape sequence **\"** stands for the double-quotation mark character. If you want to escape this sequence of characters, use a preceding backslash, like this: **\\"**. To summarize: If you need to write a file path that contains spaces or an end backslash, write it like this: **"C:\My Directory\ \"**.

Options

Options are listed in short form (if available) and long form. You can use one or two dashes for both short and long forms. An option may or may not take a value. If it takes a value, it is written like this: **--option=value**. 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. If an option takes a Boolean value and no value is specified, then the option's default value is `TRUE`. Use the **--h, --help** option to display information about the command.

▼ XML to JSON conversion options

 © 2017-2023 Altova GmbH

These options define the handling of specific conversion-related details in conversions between XML and JSON.

- attributes

  --attributes = true|false
  If set to **true**, then conversion between XML attributes and JSON @-prefixed properties occurs. Otherwise, XML attributes and JSON @-properties will not be converted. The default is **true**.
  Note: Boolean option values are set to true if the option is specified without a value.

- comments

  --comments = true|false
  If set to **true**, then conversion between XML comments and JSON #-prefixed properties occurs. Otherwise, XML attributes and JSON #-properties will not be converted. The default is **true**.
  Note: Boolean option values are set to true if the option is specified without a value.

- conversion-output, o

  --o, --converson-output = FILE
  Sets the path and name of the file to which the result of the conversion is sent.

- ignore-pis

  --ignore-pis = true|false
  If set to **true**, ignores processing instructions in the source XML document. The default is **false**.
  Note: Boolean option values are set to true if the option is specified without a value.

- merge-elements

  --merge-elements = true|false
  If set to **true**, creates an array in the generated JSON document from same-name, same-level elements in the XML document. The default is **true**.
  Note: Boolean option values are set to true if the option is specified without a value.

- merge-text

  --merge-text = true|false
  If set to **true**, creates an array in the generated JSON document from same-level text nodes in the XML document. The default is **true**.
  Note: Boolean option values are set to true if the option is specified without a value.

- pi

  --pi = true|false
  If set to **true**, then conversion between XML processing instructions and JSON ?-prefixed properties occurs. Otherwise, XML attributes and JSON ?-properties will not be converted. The default is **true**.
  Note: Boolean option values are set to true if the option is specified without a value.

- pretty-print

  --pp, --pretty-print = true|false
  If set to **true**, pretty-prints the generated output document. The default is **false**.
  Note: Boolean option values are set to true if the option is specified without a value.

- text

  --text = `true|false`
  If set to **true**, then conversion between XML text content and JSON $-prefixed properties occurs. Otherwise, XML attributes and JSON $-properties will not be converted. The default is **true**.
  Note: Boolean option values are set to `true` if the option is specified without a value.

- Common options
  - 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 | unlimited*
    Specifies the error limit with a value range of `1` to `9999` or `unlimited`. The default value is `100`. Processing stops when the error limit is reached. Useful for limiting processor use during validation/transformation.

  - info-limit

    --info-limit = *N | unlimited*
    Specifies the information message limit in the range `1-65535` or `unlimited`. Processing continues if the specified info limit is reached, but further messages are not reported. The default value is `100`.

  - 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`.)

  - 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.

  - log-output

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

  - network-timeout

    --network-timeout = *VALUE*
    Specifies the timeout in milliseconds for remote I/O operations. Default is: `40000`.

 © 2017-2023 Altova GmbH

- recurse

  --recurse = `true|false`
  Used to select files within sub-directories, including in ZIP archives. 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. References to ZIP files must be given in quotes. The wildcard characters `*` and `?` may be used. So, `*.xml` will select all `.xml` files in the (zip) folder. The option's default value is `false`.
  Note: Boolean option values are set to `true` if the option is specified without a value.

- 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 Server. If used with a command, place `--version` before the command.

- warning-limit

  --warning-limit = *N | unlimited*
  Specifies the warning limit in the range `1-65535` or `unlimited`. Processing continues if this limit is reached, but further warnings are not reported. The default value is `100`.

- 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\RaptorXMLServer2024\etc\RootCatalog.xml`). See the section, XML Catalogs 47, 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 47, for information about working with catalogs.

  - enable-globalresources

    --enable-globalresources = `true|false`

Enables global resources[53]. 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[53] (and enables global resources)[53].

- globalresourcefile [gr]

  `--gr | --globalresourcefile =` *`FILE`*
  Specifies the global resource file[53] (and enables global resources)[53].

5.5.11 xsd2jsonschema

The **`xsd2jsonschema`** command converts one or more W3C XML Schema 1.0 or 1.1 documents to a JSON Schema document.

```
raptorxml xsd2jsonschema [options] XSDFile
```

- The *`XSDFile`* argument is the XML Schema file to convert.
- Use the `--schema-conversion-output` option to specify the location of the generated XSD file.

Example

Example of the **`xsd2jsonschema`** command:

- **`raptorxml`** `xsd2jsonschema --schema-conversion-output=c:\MyJSONSchema.json c:\MyXSDSchema.xsd`

- Casing and slashes on the command line

  `RaptorXML` (and **`RaptorXMLServer`** for administration commands) *on Windows*
  `raptorxml` (and **`raptorxmlserver`** for administration commands) *on Windows and Unix (Linux, Mac)*

  * Note that lowercase (`raptorxml` and `raptorxmlserver`) works on all platforms (Windows, Linux, and Mac), while upper-lower (`RaptorXML`) works only on Windows and Mac.
  * Use forward slashes on Linux and Mac, backslashes on Windows.

- Backslashes, spaces, and special characters on Windows systems

  On Windows systems: When spaces or special characters occur in strings (for example in file or folder names, or company, person or product names), use quotes: for example, **`"My File"`**. Note, however, that a backslash followed by a double-quotation mark (for example, **`"C:\My directory\"`**) might not be read correctly. This is because the backslash character is also used to indicate the start of an escape sequence, and the escape sequence **`\"`** stands for the double-quotation mark character. If you want to

 © 2017-2023 Altova GmbH

escape this sequence of characters, use a preceding backslash, like this: `\\"`. To summarize: If you need to write a file path that contains spaces or an end backslash, write it like this: `"C:\My Directory\\"`.

Options

Options are listed in short form (if available) and long form. You can use one or two dashes for both short and long forms. An option may or may not take a value. If it takes a value, it is written like this: **`--option=value`**. 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. If an option takes a Boolean value and no value is specified, then the option's default value is `TRUE`. Use the **`--h, --help`** option to display information about the command.

- XML Schema definition options
  - 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 option specifies whether to load a schema document or just license a namespace, and, if a schema document is to be loaded, which information should be used to find it. Default: `load-preferring-schemalocation`.
    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 [47]. 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 [47]. If no `schemaLocation` attribute is present, then the value of the `namespace` attribute is used via a catalog mapping [47]. This is the **default value**.
    - `load-by-namespace`: The value of the `namespace` attribute is used to locate the schema via a catalog mapping [47].
    - `load-combining-both`: If either the `namespace` or `schemaLocation` attribute has a catalog mapping [47], then the mapping is used. If both have catalog mappings [47], then the value of the `--schema-mapping` option (XML/XSD option [225]) decides which mapping is used. If no catalog mapping [47] 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 schema location and namespace are both used to find a schema document, specifies which of them should be preferred during catalog lookup. (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 [47], 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
Specifies the behavior of the xsi:schemaLocation and xsi:noNamespaceSchemaLocation attributes: Whether to load a schema document, and, if yes, which information should be used to find it. Default: load-by-schemalocation.

- The load-by-schemalocation value uses the URL of the schema location[383] in the xsi:schemaLocation and xsi:noNamespaceSchemaLocation attributes in XML instance documents. This is the **default value**.
- The load-by-namespace value takes the namespace part[383] of xsi:schemaLocation and an empty string in the case of xsi:noNamespaceSchemaLocation and locates the schema via a catalog mapping[47].
- If load-combining-both is used and if either the namespace part or the URL part has a catalog mapping[47], then the catalog mapping[47] is used. If both have catalog mappings[47], then the value of the --schema-mapping option (XML/XSD option[225]) 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.

▼ XMLSchema processing options

▼ report-import-namespace-mismatch-as-warning

--report-import-namespace-mismatch-as-warning = true|false
Downgrades namespace or target-namespace mismatch errors when importing schemas with **xs:import** from errors to warnings. The 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-for-schemas

--xml-mode-for-schemas = wf|id|valid
Specifies the XML processing mode to use for XML schema documents: wf=wellformed check; id=wellformed with ID/IDREF checks; valid=validation. Default value is wf. Note that a value of valid requires that each schema document loaded during processing references a DTD. If no DTD exists, an error is reported.

▼ 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

 © 2017-2023 Altova GmbH

is detected as being version `1.0`.

- Conversion from XSD to JSON Schema
  - array-and-item

    `--array-and-item =` `true|false`
    If set to **`true`**, the generated JSON Schema will permit not only arrays but also single items for particles with `maxOccurs > 1`. The default is **`true`**.
    Note: Boolean option values are set to `true` if the option is specified without a value.

  - jsonschema-version

    `--jsonschema-version =` `draft04|draft06|draft07|2019-09|2020-12|latest|detect`
    Specifies which version of the JSON Schema specification draft version to use. Default is `detect`.

  - property-for-comments

    `--property-for-comments =` `true|false`
    If set to **`true`**, creates a property named `'#'` in each sub-schema to support comments. The default is **`false`**.
    Note: Boolean option values are set to `true` if the option is specified without a value.

  - property-for-pis

    `--property-for-pis =` `true|false`
    If set to **`true`**, creates a pattern property matching properties prefixed with `'?'` to support XML processing instructions. The default is **`false`**.
    Note: Boolean option values are set to `true` if the option is specified without a value.

  - property-for-xmlns

    `--property-for-xmlns =` `true|false`
    If set to **`true`**, creates a pattern property matching properties prefixed with `'@xmlns'` in each sub-schema to support namespace declaration. The default is **`false`**.
    Note: Boolean option values are set to `true` if the option is specified without a value.

  - property-for-xsi

    `--property-for-xsi =` `true|false`
    If set to **`true`**, creates a pattern property matching properties prefixed with `'@xsi'` in each sub-schema to support `xsi:*` attributes, such as `xsi:schemaLocation`. The default is **`false`**.
    Note: Boolean option values are set to `true` if the option is specified without a value.

  - simple-content-pure-object

    `--simple-content-pure-object=` `true|false`
    If set to **`true`**, creates a pure object for complex types with simple content. The default is **`false`**.
    Note: Boolean option values are set to `true` if the option is specified without a value.

  - schema-conversion-output, o

    `--o, --schema-converson-output =` `FILE`

Sets the path and name of the file to which the result of the conversion is sent.

- simplify-occurrence-constraints

  --simplify-occurrence-constraints = true|false
  If set to **true**: (i) occurrence specifications in the XML Schema are simplified to either required or optional in the JSON Schema: (ii) repeatable elements in the XML Schema are simplified to arrays with unbounded **maxItems**. The default is **false**.
  Note: Boolean option values are set to true if the option is specified without a value.

- xmlns

  --xml-mode = wf|id|valid
  Specifies prefix URI mappings for the namespaces in the XML Schema.

- Common options
  - 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 | unlimited*
    Specifies the error limit with a value range of 1 to 9999 or unlimited. The default value is 100. Processing stops when the error limit is reached. Useful for limiting processor use during validation/transformation.

  - info-limit

    --info-limit = *N | unlimited*
    Specifies the information message limit in the range 1-65535 or unlimited. Processing continues if the specified info limit is reached, but further messages are not reported. The default value is 100.

  - 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.)

  - 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.

  - log-output

 © 2017-2023 Altova GmbH

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

- network-timeout

  --network-timeout = *VALUE*
  Specifies the timeout in milliseconds for remote I/O operations. Default is: `40000`.

- recurse

  --recurse = `true|false`
  Used to select files within sub-directories, including in ZIP archives. 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. References to ZIP files must be given in quotes. The wildcard characters `*` and `?` may be used. So, `*.xml` will select all `.xml` files in the (zip) folder. The option's default value is `false`.
  Note: Boolean option values are set to `true` if the option is specified without a value.

- 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 Server. If used with a command, place `--version` before the command.

- warning-limit

  --warning-limit = *N | unlimited*
  Specifies the warning limit in the range `1-65535` or `unlimited`. Processing continues if this limit is reached, but further warnings are not reported. The default value is `100`.

- 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\RaptorXMLServer2024\etc\RootCatalog.xml`). See the section, XML Catalogs [47], 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 [47], for information about working with catalogs.

- enable-globalresources

  --enable-globalresources = `true|false`
  Enables global resources [53]. 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 [53] (and enables global resources) [53].

- globalresourcefile [gr]

  --gr | --globalresourcefile = *FILE*
  Specifies the global resource file [53] (and enables global resources) [53].

© 2017-2023 Altova GmbH

5.6 XML Signature Commands

The XML Signature commands can be used to sign an XML document and to verify a signed document. These commands are listed below and described in detail in the sub-sections of this section:

- xmlsignature-sign 183: Creates an XML signature output document from an input document
- xmlsignature-verify 187: Verifies an XML signature document
- xmlsignature-update 183: Updates the signature of a (modified) XML document
- xmlsignature-remove 183: Removes the signature of an XML document

5.6.1 xmlsignature-sign

The **xmlsignature-sign | xsign** command takes an XML document as input and creates an XML signature output document using the specified signing options.

```
raptorxml xmlsignature-sign [options] --output=File --signature-type=Value --
signature-canonicalization-method=Value --certname=Value|hmackey=Value InputFile
```

- The *InputFile* argument is the XML document to sign.
- The **--output** option specifies the location of the document that contains the XML signature.

Example

Example of the **xmlsignature-sign** command:

- **raptorxml** xsign --output=c:\SignedFile.xml --signature-type=enveloped --signature-canonicalization-method=xml-c14n11 --hmackey=secretpassword c:\SomeUnsigned.xml

▼ Casing and slashes on the command line

RaptorXML (and **RaptorXMLServer** for administration commands) *on Windows*
raptorxml (and **raptorxmlserver** for administration commands) *on Windows and Unix (Linux, Mac)*

* Note that lowercase (raptorxml and raptorxmlserver) works on all platforms (Windows, Linux, and Mac), while upper-lower (RaptorXML) works only on Windows and Mac.
* Use forward slashes on Linux and Mac, backslashes on Windows.

▼ Backslashes, spaces, and special characters on Windows systems

On Windows systems: When spaces or special characters occur in strings (for example in file or folder names, or company, person or product names), use quotes: for example, **"My File"**. Note, however, that a backslash followed by a double-quotation mark (for example, **"C:\My directory\"**) might not be read correctly. This is because the backslash character is also used to indicate the start of an escape sequence, and the escape sequence **\"** stands for the double-quotation mark character. If you want to escape this sequence of characters, use a preceding backslash, like this: **\\"**. To summarize: If you need to write a file path that contains spaces or an end backslash, write it like this: **"C:\My Directory\ \"**.

Options

Options are listed in short form (if available) and long form. You can use one or two dashes for both short and long forms. An option may or may not take a value. If it takes a value, it is written like this: **--option=value**. 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. If an option takes a Boolean value and no value is specified, then the option's default value is `TRUE`. Use the **--h, --help** option to display information about the command.

Common options

output

output = *FILE*
The URL of the output document that is created with the new XML signature.

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.

XML Signature options

absolute-reference-uri

--absolute-reference-uri = `true|false`
Specifies whether the URI of the signed document is to be read as absolute (`true`) or relative (`false`). Default is `false`.
Note: Boolean option values are set to `true` if the option is specified without a value.

certname, certificate-name

--certname, --certificate-name = *VALUE*
The name of the certificate used for signing.

Windows
This is the **Subject** name of a certificate from the selected **--certificate-store**.

Example to list the certificates (under PowerShell)

```
% ls cert://CurrentUser/My
PSParentPath: Microsoft.PowerShell.Security\Certificate::CurrentUser\My
Thumbprint Subject
---------- -------
C9DF64BB0AAF5FA73474D78B7CCFFC37C95BFC6C CN=certificate1
... CN=...
```

Example: **--certificate-name==certificate1**

Linux/MacOS

© 2017-2023 Altova GmbH

--certname specifies the file name of a PEM encoded X.509v3 certificate with the private key. Such files usually have the extension **.pem**.

xample: **--certificate-name==/path/to/certificate1.pem**

- certstore, certificate-store

  --certstore, --certificate-store = *VALUE*
  The location where the certificate specified with **--certificate-name** is stored.

  Windows
  The name of a certificate store under **cert://CurrentUser**. The available certificate stores can be listed (under PowerShell) by using **% ls cert://CurrentUser/**. Certificates would then be listed as follows:

```
Name : TrustedPublisher
Name : ClientAuthIssuer
Name : Root
Name : UserDS
Name : CA
Name : ACRS
Name : REQUEST
Name : AuthRoot
Name : MSIEHistoryJournal
Name : TrustedPeople
Name : MyCertStore
Name : Local NonRemovable Certificates
Name : SmartCardRoot
Name : Trust
Name : Disallowed
```

  Example: **--certificate-store==MyCertStore**

  Linux/MacOS
  The **--certstore** option is currently not supported.

- digest, digest-method

  --digest, --digest-method = sha1|sha256|sha384|sha512
  The algorithm that is used to compute the digest value over the input XML file. Available values are: sha1|sha256|sha384|sha512.

- hmackey, hmac-secret-key

  --hmackey, --hmac-secret-key = *VALUE*
  The HMAC shared secret key; must have a minimum length of six characters.

  Example: **--hmackey=secretpassword**

- hmaclen, hmac-output-length

  --hmaclen, --hmac-output-length = *LENGTH*

Truncates the output of the HMAC algorithm to `length` bits. If specified, this value must be

- a multiple of 8
- larger than 80
- larger than half of the underlying hash algorithm's output length

▼ keyinfo, append-keyinfo

`--keyinfo, --append-keyinfo =` `true|false`
Specifies whether to include the `KeyInfo` element in the signature or not. The default is `false`.

▼ sigc14nmeth, signature-canonicalization-method

`--sigc14nmeth, --signature-canonicalization-method =` *VALUE*
Specifies the canonicalization algorithm to apply to the **`SignedInfo`** element. The value must be one of:

- `REC-xml-c14n-20010315`
- `xml-c14n11`
- `xml-exc-c14n#`

▼ sigmeth, signature-method

`--sigmeth, --signature-method =` *VALUE*
Specifies the algorithm to use for generating the signature.

When a certificate is used
If a certificate is specified, then `SignatureMethod` is optional and the value for this parameter is derived from the certificate. If specified, it must match the algorithm used by the certificate. Example: `rsa-sha256`.

When --hmac-secret-key is used
When `HMACSecretKey` is used, then `SignatureMethod` is mandatory. The value must be one of the supported HMAC algorithms:

- `hmac-sha256`
- `hmac-sha386`
- `hmac-sha512`
- `hmac-sha1` (discouraged by the specification)

Example: **`hmac-sha256`**

▼ sigtype, signature-type

`--sigtype, --signature-type =` `detached | enveloping | enveloped`
Specifies the type of signature to be generated.

▼ transforms

`--transforms =` *VALUE*
Specifies the XML Signature transformations applied to the input document. The supported values are:

- `REC-xml-c14n-20010315` for Canonical XML 1.0 (omit comments)

 © 2017-2023 Altova GmbH

- `xml-c14n11` for Canonical XML 1.1 (omit comments)
- `xml-exc-c14n#` for Exclusive XML Canonicalization 1.0 (omit comments)
- `REC-xml-c14n-20010315#WithComments` for Canonical XML 1.0 (with comments)
- `xml-c14n11#WithComments` for Canonical XML 1.1 (with comments)
- `xml-exc-c14n#WithComments` for Exclusive XML Canonicalization 1.0 (with comments)
- `base64`
- `strip-whitespaces` Altova extension

Example: **`--transforms=xml-c14n11`**

Note: This option can be specified multiple times. If specified multiple times, then the order of specification is significant. The first specified transformation receives the input document. The last specified transformation is used immediately before calculation of the digest value.

▼ write-default-attributes

`--write-default-attributes =` `true|false`
Specifies whether to include default attribute values from the DTD in the signed document.

▼ Help and version options

▼ 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`.)

▼ version

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

5.6.2 xmlsignature-verify

The **`xmlsignature-verify | xverify`** command verifies the XML signature of the input file.

```
raptorxml xmlsignature-verify [options] InputFile
```

- The *`InputFile`* argument is the signed XML document to verify.
- If the verification is successful, a **`result="OK"`** message is displayed; otherwise, a **`result="Failed"`** message is displayed.

Example

Example of the **`xmlsignature-verify`** command:

- **raptorxml** xverify c:\SignedFile.xml

▼ Casing and slashes on the command line

RaptorXML (and **RaptorXMLServer** for administration commands) *on Windows*
raptorxml (and **raptorxmlserver** for administration commands) *on Windows and Unix (Linux, Mac)*

* Note that lowercase (raptorxml and raptorxmlserver) works on all platforms (Windows, Linux, and Mac), while upper-lower (RaptorXML) works only on Windows and Mac.
* Use forward slashes on Linux and Mac, backslashes on Windows.

▼ Backslashes, spaces, and special characters on Windows systems

On Windows systems: When spaces or special characters occur in strings (for example in file or folder names, or company, person or product names), use quotes: for example, **"My File"**. Note, however, that a backslash followed by a double-quotation mark (for example, **"C:\My directory\"**) might not be read correctly. This is because the backslash character is also used to indicate the start of an escape sequence, and the escape sequence **\"** stands for the double-quotation mark character. If you want to escape this sequence of characters, use a preceding backslash, like this: **\\"**. To summarize: If you need to write a file path that contains spaces or an end backslash, write it like this: **"C:\My Directory\ \"**.

Options

Options are listed in short form (if available) and long form. You can use one or two dashes for both short and long forms. An option may or may not take a value. If it takes a value, it is written like this: **--option=value**. 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. If an option takes a Boolean value and no value is specified, then the option's default value is TRUE. Use the **--h, --help** option to display information about the command.

▼ Common options

▼ 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.

▼ XML Signature options

▼ certname, certificate-name

--certname, --certificate-name = *VALUE*
The name of the certificate used for signing.

Windows
This is the **Subject** name of a certificate from the selected **--certificate-store**.

Example to list the certificates (under PowerShell)

 © 2017-2023 Altova GmbH

```
% ls cert://CurrentUser/My
PSParentPath: Microsoft.PowerShell.Security\Certificate::CurrentUser\My
Thumbprint Subject
---------- -------
C9DF64BB0AAF5FA73474D78B7CCFFC37C95BFC6C CN=certificate1
... CN=...
```

Example: **--certificate-name==certificate1**

Linux/MacOS

--certname specifies the file name of a PEM encoded X.509v3 certificate with the private key. Such files usually have the extension **.pem**.

xample: **--certificate-name==/path/to/certificate1.pem**

- certstore, certificate-store

  --certstore, --certificate-store = *VALUE*

  The location where the certificate specified with **--certificate-name** is stored.

  Windows

  The name of a certificate store under **cert://CurrentUser**. The available certificate stores can be listed (under PowerShell) by using **% ls cert://CurrentUser/**. Certificates would then be listed as follows:

```
Name : TrustedPublisher
Name : ClientAuthIssuer
Name : Root
Name : UserDS
Name : CA
Name : ACRS
Name : REQUEST
Name : AuthRoot
Name : MSIEHistoryJournal
Name : TrustedPeople
Name : MyCertStore
Name : Local NonRemovable Certificates
Name : SmartCardRoot
Name : Trust
Name : Disallowed
```

  Example: **--certificate-store==MyCertStore**

  Linux/MacOS

  The **--certstore** option is currently not supported.

- hmackey, hmac-secret-key

  --hmackey, --hmac-secret-key = *VALUE*

  The HMAC shared secret key; must have a minimum length of six characters.

Example: **`--hmackey=secretpassword`**

- Help and version options
  - 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`.)

  - version

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

5.6.3 xmlsignature-update

The **`xmlsignature-update | xupdate`** command updates the XML signature in the signed input file. If the document has been modified, the updated XML signature will be different; otherwise, the updated signature will be the same as the previous signature.

```
raptorxml xmlsignature-update [options] --output=File SignedFile
```

- The *`SignedFile`* argument is the signed XML document to update.
- Either (i) the `hmac-secret-key` option or (ii) the `certificate-name` and `certificate-store` options must be specified.
- If the `certificate-name` and `certificate-store` options are specified, then they must match those that were used to sign the XML document previously. (Note that the `certificate-store` option is currently not supported on Linux and macOS.)

Examples

Examples of the **`xmlsignature-update`** command:

- **`raptorxml`** `xupdate --output=c:\UpdatedSignedFile.xml --certname=certificate1 --certstore=MyCertStore c:\SomeSignedFile.xml`
- **`raptorxml`** `xupdate --output=c:\UpdatedSignedFile.xml --hmackey=SecretPassword c:\SomeSignedFile.xml`

- Casing and slashes on the command line

  `RaptorXML` (and **`RaptorXMLServer`** for administration commands) *on Windows*
  `raptorxml` (and **`raptorxmlserver`** for administration commands) *on Windows and Unix (Linux, Mac)*

  * Note that lowercase (`raptorxml` and `raptorxmlserver`) works on all platforms (Windows, Linux, and

 © 2017-2023 Altova GmbH

Mac), while upper-lower (RaptorXML) works only on Windows and Mac.
* Use forward slashes on Linux and Mac, backslashes on Windows.

▼ Backslashes, spaces, and special characters on Windows systems

On Windows systems: When spaces or special characters occur in strings (for example in file or folder names, or company, person or product names), use quotes: for example, **"My File"**. Note, however, that a backslash followed by a double-quotation mark (for example, **"C:\My directory\"**) might not be read correctly. This is because the backslash character is also used to indicate the start of an escape sequence, and the escape sequence **\"** stands for the double-quotation mark character. If you want to escape this sequence of characters, use a preceding backslash, like this: **\\"**. To summarize: If you need to write a file path that contains spaces or an end backslash, write it like this: **"C:\My Directory\ \"**.

Options

Options are listed in short form (if available) and long form. You can use one or two dashes for both short and long forms. An option may or may not take a value. If it takes a value, it is written like this: **--option=value**. 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. If an option takes a Boolean value and no value is specified, then the option's default value is TRUE. Use the **--h, --help** option to display information about the command.

▼ Common options

▼ output

output = *FILE*
The URL of the output document that is created with the new XML signature.

▼ 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.

▼ XML Signature options

▼ certname, certificate-name

--certname, --certificate-name = *VALUE*
The name of the certificate used for signing.

Windows
This is the **Subject** name of a certificate from the selected **--certificate-store**.

Example to list the certificates (under PowerShell)

```
% ls cert://CurrentUser/My
PSParentPath: Microsoft.PowerShell.Security\Certificate::CurrentUser\My
Thumbprint Subject
---------- -------
```

```
C9DF64BB0AAF5FA73474D78B7CCFFC37C95BFC6C CN=certificate1
... CN=...
```

Example: **--certificate-name==certificate1**

Linux/MacOS
--certname specifies the file name of a PEM encoded X.509v3 certificate with the private key. Such files usually have the extension **.pem**.

xample: **--certificate-name==/path/to/certificate1.pem**

- certstore, certificate-store

  --certstore, --certificate-store = *VALUE*
  The location where the certificate specified with **--certificate-name** is stored.

  Windows
  The name of a certificate store under **cert://CurrentUser**. The available certificate stores can be listed (under PowerShell) by using **% ls cert://CurrentUser/**. Certificates would then be listed as follows:

```
Name : TrustedPublisher
Name : ClientAuthIssuer
Name : Root
Name : UserDS
Name : CA
Name : ACRS
Name : REQUEST
Name : AuthRoot
Name : MSIEHistoryJournal
Name : TrustedPeople
Name : MyCertStore
Name : Local NonRemovable Certificates
Name : SmartCardRoot
Name : Trust
Name : Disallowed
```

  Example: **--certificate-store==MyCertStore**

  Linux/MacOS
  The **--certstore** option is currently not supported.

- hmackey, hmac-secret-key

  --hmackey, --hmac-secret-key = *VALUE*
  The HMAC shared secret key; must have a minimum length of six characters.

  Example: **--hmackey=secretpassword**

 © 2017-2023 Altova GmbH

- Help and version options
  - 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`.)

  - version

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

5.6.4 xmlsignature-remove

The **xmlsignature-remove | xremove** command removes the XML signature of the signed input file, and saves the resulting unsigned document to an output location that you specify.

```
raptorxml xmlsignature-remove [options] --output=File SignedFile
```

- The *SignedFile* argument is the signed XML document from which you want to remove the XML signature.
- The `--output` option specifies the location of the unsigned XML document that is generated.

Example

Example of the **xmlsignature-remove** command:

- **raptorxml** `xremove --output=c:\UnsignedFile.xml c:\SignedFile.xml`

- Casing and slashes on the command line

  RaptorXML (and **RaptorXMLServer** for administration commands) *on Windows*
  raptorxml (and **raptorxmlserver** for administration commands) *on Windows and Unix (Linux, Mac)*

  * Note that lowercase (`raptorxml` and `raptorxmlserver`) works on all platforms (Windows, Linux, and Mac), while upper-lower (`RaptorXML`) works only on Windows and Mac.
  * Use forward slashes on Linux and Mac, backslashes on Windows.

- Backslashes, spaces, and special characters on Windows systems

  On Windows systems: When spaces or special characters occur in strings (for example in file or folder names, or company, person or product names), use quotes: for example, **"My File"**. Note, however, that a backslash followed by a double-quotation mark (for example, **"C:\My directory\"**) might not be read correctly. This is because the backslash character is also used to indicate the start of an escape sequence, and the escape sequence **\"** stands for the double-quotation mark character. If you want to escape this sequence of characters, use a preceding backslash, like this: **\\"**. To summarize: If you

need to write a file path that contains spaces or an end backslash, write it like this: `"C:\My Directory\ \"`.

Options

Options are listed in short form (if available) and long form. You can use one or two dashes for both short and long forms. An option may or may not take a value. If it takes a value, it is written like this: **`--option=value`**. 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. If an option takes a Boolean value and no value is specified, then the option's default value is `TRUE`. Use the **`--h, --help`** option to display information about the command.

- Common options
  - output

    `output =` *FILE*
    The URL of the output document that is created with the XML signature removed.

  - 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.

- Help and version options
  - 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`.)

  - version

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

© 2017-2023 Altova GmbH

5.7 General Commands

This section contains a description of the following general commands:

- valany 195: validates the submitted document according to its type
- script 196: executes a Python script
- help 197: displays information about the named command

5.7.1 valany

The **valany** command is a general command that validates a document on the basis of what type of document it is. The type of the input document is detected automatically, and the corresponding validation is carried out according to the respective specification. The *InputFile* argument is the document to validate. Note that only one document can be submitted as the argument of the command.

```
raptorxml valany [options] InputFile
```

The **valany** command covers the following types of validation. Its options are those that are available for the corresponding individual validation command. See the description of the respective validation commands for a list of their respective options.

- valdtd (dtd) 69
- valxsd (xsd) 73
- valxml-withdtd (xml) 58
- valxml-withxsd (xsi) 62
- valxslt 129
- valxquery 108
- valavrojson (avrojson) 152

Examples

- **raptorxml** valany c:\Test.xsd

▼ Casing and slashes on the command line

RaptorXML (and **RaptorXMLServer** for administration commands) *on Windows*
raptorxml (and **raptorxmlserver** for administration commands) *on Windows and Unix (Linux, Mac)*

* Note that lowercase (raptorxml and raptorxmlserver) works on all platforms (Windows, Linux, and Mac), while upper-lower (RaptorXML) works only on Windows and Mac.
* Use forward slashes on Linux and Mac, backslashes on Windows.

▼ Backslashes, spaces, and special characters on Windows systems

On Windows systems: When spaces or special characters occur in strings (for example in file or folder names, or company, person or product names), use quotes: for example, **"My File"**. Note, however, that a backslash followed by a double-quotation mark (for example, **"C:\My directory\"**) might not be read

correctly. This is because the backslash character is also used to indicate the start of an escape sequence, and the escape sequence `\"` stands for the double-quotation mark character. If you want to escape this sequence of characters, use a preceding backslash, like this: `\\"`. To summarize: If you need to write a file path that contains spaces or an end backslash, write it like this: `"C:\My Directory\ \"`.

Options

See the description of the respective validation commands for a list of their respective options. Note, however, that while most individual validation commands accept multiple input documents, the **valany** command accepts only one input document. Options such as the `--listfile` option will therefore not apply to **valany**.

5.7.2 script

The **script** command executes a Python 3.11.5 script that uses the RaptorXML Python API.

```
raptorxml script [options] PythonScriptFile
```

The *File* argument is the path to the Python script you want to execute. Additional options are available for this command. To obtain a list of these options, run the following command:

```
raptorxml script [-h | --help]
```

Examples

- **raptorxml** script c:\MyPythonScript.py
- **raptorxml** script -h
- **raptorxml** script # *Without a script file, an interactive Python shell is started*
- **raptorxml** script -m pip # *Loads and executes the* pip *module; see the Options section below*

▼ Casing and slashes on the command line

RaptorXML (and **RaptorXMLServer** for administration commands) *on Windows*
raptorxml (and **raptorxmlserver** for administration commands) *on Windows and Unix (Linux, Mac)*

* Note that lowercase (raptorxml and raptorxmlserver) works on all platforms (Windows, Linux, and Mac), while upper-lower (RaptorXML) works only on Windows and Mac.
* Use forward slashes on Linux and Mac, backslashes on Windows.

▼ Backslashes, spaces, and special characters on Windows systems

On Windows systems: When spaces or special characters occur in strings (for example in file or folder names, or company, person or product names), use quotes: for example, **"My File"**. Note, however, that a backslash followed by a double-quotation mark (for example, **"C:\My directory\"**) might not be read correctly. This is because the backslash character is also used to indicate the start of an escape

 © 2017-2023 Altova GmbH

sequence, and the escape sequence `\"` stands for the double-quotation mark character. If you want to escape this sequence of characters, use a preceding backslash, like this: `\\"`. To summarize: If you need to write a file path that contains spaces or an end backslash, write it like this: `"C:\My Directory\ \"`.

Options

Any options and arguments after the `script` command are forwarded directly to the Python interpreter. Please consult the Python documentation page https://docs.python.org/3.11/using/cmdline.html for a complete listing of available options.

5.7.3 help

Syntax and description

The **`help`** command takes a single argument (`Command`), which is the name of the command for which help is required. It displays the command's syntax, its options, and other relevant information. If the `Command` argument is not specified, then all commands of the executable are listed, with each having a brief text description. The **`help`** command can be called from either executable: **`raptorxml`** or **`raptorxmlserver`**.

```
raptorxml help Command
raptorxmlserver help Command
```

- Casing and slashes on the command line

  `RaptorXML` (and **`RaptorXMLServer`** for administration commands) *on Windows*
  `raptorxml` (and **`raptorxmlserver`** for administration commands) *on Windows and Unix (Linux, Mac)*

  * Note that lowercase (`raptorxml` and `raptorxmlserver`) works on all platforms (Windows, Linux, and Mac), while upper-lower (`RaptorXML`) works only on Windows and Mac.
  * Use forward slashes on Linux and Mac, backslashes on Windows.

Example

Examples of the **`help`** command to display information about the **`licenserver`** command (this command is available in both executables):

```
raptorxml help licenseserver
raptorxmlserver help licenseserver
```

The --help option

Help information about a command is also available by using the `--help` option of the command for which help information is required. The two commands below produce the same results:

```
raptorxml licenseserver --help
```

The command above uses the `--help` option of the **`licenseserver`** command.

```
raptorxml help licenseserver
```

The **`help`** command takes `licenseserver` as its argument.

Both commands display help information about the **`licenseserver`** command.

© 2017-2023 Altova GmbH

5.8 Localization Commands

You can create a localized version of the RaptorXML application for any language of your choice. Five localized versions (English, German, Spanish, French, and Japanese) are already available in the `<ProgramFilesFolder>\Altova\RaptorXMLServer2024\bin\` folder. These five 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` [199] command. The resource strings in the generated XML file will be one of the five supported languages: English (`en`), German (`de`), Spanish (`es`), French (`fr`), 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\RaptorXMLServer2024\bin\` folder. Your DLL file will have a name of the form `RaptorXMLServer_lc.dll`. The `_lc` part of the name contains the language code. For example, in `RaptorXMLServer_de.dll`, the `de` part is the language code for German (Deutsch).
5. Run the `setdeflang` [201] command to set your localized DLL file as the RaptorXML application to use. For the argument of the `setdeflang` [201] command, use the language code that is part of the DLL name.

Note: Altova RaptorXML Server is delivered with support for five languages: English, German, Spanish, French, and Japanese. So you do not need to create a localized version of these languages. To set any of these five languages as the default language, use the CLI's `setdeflang` [201] command.

5.8.1 exportresourcestrings

Syntax and description

The **`exportresourcestrings`** command outputs an XML file containing the resource strings of the RaptorXML Server application in the specified language. Available export languages are English (`en`), German (`de`), Spanish (`es`), French (`fr`), and Japanese (`ja`).

```
raptorxml exportresourcestrings [options] LanguageCode XMLOutputFile
raptorxmlserver exportresourcestrings [options] LanguageCode XMLOutputFile
```

- The ***`LanguageCode`*** argument gives the language of the resource strings in the output XML file; this is the *export language*. Allowed export languages (with their language codes in parentheses) are: English (`en`), German, (`de`), Spanish (`es`), French (`fr`), and Japanese (`ja`).
- The ***`XMLOutputFile`*** argument specifies the path and name of the output XML file.
- The **`exportresourcestrings`** command can be called from either executable: **`raptorxml`** or **`raptorxmlserver`**.

How to create localizations is described below.

- Casing and slashes on the command line

  RaptorXML (and **RaptorXMLServer** for administration commands) *on Windows*
  raptorxml (and **raptorxmlserver** for administration commands) *on Windows and Unix (Linux, Mac)*

  * Note that lowercase (`raptorxml` and `raptorxmlserver`) works on all platforms (Windows, Linux, and Mac), while upper-lower (`RaptorXML`) works only on Windows and Mac.
  * Use forward slashes on Linux and Mac, backslashes on Windows.

- Backslashes, spaces, and special characters on Windows systems

  On Windows systems: When spaces or special characters occur in strings (for example in file or folder names, or company, person or product names), use quotes: for example, `"My File"`. Note, however, that a backslash followed by a double-quotation mark (for example, `"C:\My directory\"`) might not be read correctly. This is because the backslash character is also used to indicate the start of an escape sequence, and the escape sequence `\"` stands for the double-quotation mark character. If you want to escape this sequence of characters, use a preceding backslash, like this: `\\"`. To summarize: If you need to write a file path that contains spaces or an end backslash, write it like this: `"C:\My Directory\ \"`.

Examples

Examples of the **exportresourcestrings** command:

```
raptorxml exportresourcestrings de c:\Strings.xml
raptorxmlserver exportresourcestrings de c:\Strings.xml
```

- The first command above creates a file called `Strings.xml` at `c:\` that contains the resource strings of RaptorXML Server in German.
- The second command calls the server-executable to do the same thing as the first example.

Creating localized versions of RaptorXML Server

You can create a localized version of RaptorXML Server for any language of your choice. Five localized versions (English, German, Spanish, French, and Japanese) are already available in the `C:\Program Files (x86)\Altova\RaptorXMLServer2024\bin` folder, and therefore do not need to be created.

Create a localized version as follows:

1. Generate an XML file containing the resource strings by using the `exportresourcestrings` command (*see command syntax above*). The resource strings in this XML file will be one of the five supported languages: English (`en`), German (`de`), Spanish (`es`), French (`fr`), or Japanese (`ja`), according to the ***LanguageCode*** argument used with the command.
2. Translate the resource strings from one of the five supported languages 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 Server DLL file from your translated XML file.
4. After you receive your localized DLL file from Altova Support, save the DLL in the `C:\Program Files (x86)\Altova\RaptorXMLServer2024\bin` folder. Your DLL file will have a name of the form `RaptorXML2024_lc.dll`. The `_lc` part of the name contains the language code. For example, in

 © 2017-2023 Altova GmbH

RaptorXML2024_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 Server application to use. For the argument of the setdeflang command, use the language code that is part of the DLL name.

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

5.8.2 setdeflang

Syntax and description

The **setdeflang** command (short form is **sdl**) sets the default language of RaptorXML Server. Available languages are English (en), German (de), Spanish (es), French (fr), and Japanese (ja). The command takes a mandatory *LanguageCode* argument.

```
raptorxml setdeflang [options] LanguageCode
raptorxmlserver setdeflang [options] LanguageCode
```

- The *LanguageCode* argument is required and sets the default language of RaptorXML Server. The respective values to use are: en, de, es, fr, ja.
- The **setdeflang** command can be called from either executable: **raptorxml** or **raptorxmlserver**.
- Use the **--h, --help** option to display information about the command.

▼ Casing and slashes on the command line

RaptorXML (and **RaptorXMLServer** for administration commands) *on Windows*
raptorxml (and **raptorxmlserver** for administration commands) *on Windows and Unix (Linux, Mac)*

* Note that lowercase (raptorxml and raptorxmlserver) works on all platforms (Windows, Linux, and Mac), while upper-lower (RaptorXML) works only on Windows and Mac.
* Use forward slashes on Linux and Mac, backslashes on Windows.

Examples

Examples of the **setdeflang** (**sdl**) command:

```
raptorxml sdl de
raptorxml setdeflang es
raptorxmlserver setdeflang es
```

- The first command sets the default language of RaptorXML Server to German.
- The second command sets the default language of RaptorXML Server to Spanish.
- The third command is the same as the second command, but is executed by the server-executable.

Options

Use the **--h, --help** option to display information about the command.

5.9 License Commands

This section describes commands that can be used for licensing RaptorXML Server:

- licenseserver 202 to register RaptorXML Server with Altova LicenseServer on your network
- assignlicense 203 to upload a license file to LicenseServer (Windows only)
- verifylicense 205 to verify whether RaptorXML Server is licensed (Windows only)

Note: These commands can also be executed via the server executable for administration commands 207.

For more information about licensing Altova products with Altova LicenseServer, see the Altova LicenseServer documentation.

5.9.1 licenseserver

Syntax and description

The **licenseserver** command registers RaptorXML Server with the Altova LicenseServer specified by the *Server-Or-IP-Address* argument. For the licenseserver command to be executed successfully, the two servers (RaptorXML Server and LicenseServer) must be on the same network and LicenseServer must be running. You must also have administrator privileges in order to register RaptorXML Server with LicenseServer.

```
raptorxml licenseserver [options] Server-Or-IP-Address
raptorxmlserver licenseserver [options] Server-Or-IP-Address
```

- The *Server-Or-IP-Address* argument takes the name or IP address of the LicenseServer machine.
- The **licenseserver** command can be called from either executable: **raptorxml** or **raptorxmlserver**.

Once RaptorXML Server has been successfully registered with LicenseServer, you will receive a message to this effect. The message will also display the URL of the LicenseServer. You can now go to LicenseServer to assign RaptorXML Server a license. For details about licensing, see the LicenseServer documentation (https://www.altova.com/manual/en/licenseserver/3.12/).

▼ Casing and slashes on the command line

RaptorXML (and **RaptorXMLServer** for administration commands) *on Windows*
raptorxml (and **raptorxmlserver** for administration commands) *on Windows and Unix (Linux, Mac)*

* Note that lowercase (raptorxml and raptorxmlserver) works on all platforms (Windows, Linux, and Mac), while upper-lower (RaptorXML) works only on Windows and Mac.
* Use forward slashes on Linux and Mac, backslashes on Windows.

▼ Backslashes, spaces, and special characters on Windows systems

On Windows systems: When spaces or special characters occur in strings (for example in file or folder names, or company, person or product names), use quotes: for example, **"My File"**. Note, however, that a backslash followed by a double-quotation mark (for example, **"C:\My directory\"**) might not be read correctly. This is because the backslash character is also used to indicate the start of an escape sequence, and the escape sequence **\"** stands for the double-quotation mark character. If you want to

 © 2017-2023 Altova GmbH

escape this sequence of characters, use a preceding backslash, like this: `\\"`. To summarize: If you need to write a file path that contains spaces or an end backslash, write it like this: `"C:\My Directory\ \"`.

Examples

Examples of the **licenseserver** command:

```
raptorxml licenseserver DOC.altova.com
raptorxml licenseserver localhost
raptorxml licenseserver 127.0.0.1
raptorxmlserver licenseserver 127.0.0.1
```

The commands above specify, respectively, the machine named `DOC.altova.com`, and the user's machine (`localhost` and `127.0.0.1`) as the machine running Altova LicenseServer. In each case, the command registers RaptorXML Server with the LicenseServer on the machine specified. The last command calls the server-executable to execute the command.

Options

Options are listed in short form (if available) and long form. You can use one or two dashes for both short and long forms. An option may or may not take a value. If it takes a value, it is written like this: **--option=value**. 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. If an option takes a Boolean value and no value is specified, then the option's default value is `TRUE`. Use the **--h, --help** option to display information about the command.

- json [j]

  --j, --json = `true|false`

  Values are `true|false`. If `true`, prints the result of the registration attempt as a machine-parsable JSON object.

5.9.2 assignlicense (Windows only)

Syntax and description

The **assignlicense** command is available on Windows only (not on Linux or Mac systems). It uploads a license file to the Altova LicenseServer with which RaptorXML Server is registered (see the `licenseserver` command), and assigns the license to RaptorXML Server. It takes the path of a license file as its argument. The command also allows you to test the validity of a license.

```
raptorxml assignlicense [options] FILE
raptorxmlserver assignlicense [options] FILE
```

- The *FILE* argument takes the path of the license file.
- The `--test-only` option uploads the license file to LicenseServer and validates the license, but does not assign the license to RaptorXML Server.

- The **assignlicense** command can be called from either executable: **raptorxml** or **raptorxmlserver**.

For details about licensing, see the LicenseServer documentation (https://www.altova.com/manual/en/licenseserver/3.12/).

▼ Casing and slashes on the command line

RaptorXML (and **RaptorXMLServer** for administration commands) *on Windows*
raptorxml (and **raptorxmlserver** for administration commands) *on Windows and Unix (Linux, Mac)*

* Note that lowercase (raptorxml and raptorxmlserver) works on all platforms (Windows, Linux, and Mac), while upper-lower (RaptorXML) works only on Windows and Mac.
* Use forward slashes on Linux and Mac, backslashes on Windows.

▼ Backslashes, spaces, and special characters on Windows systems

On Windows systems: When spaces or special characters occur in strings (for example in file or folder names, or company, person or product names), use quotes: for example, **"My File"**. Note, however, that a backslash followed by a double-quotation mark (for example, **"C:\My directory\"**) might not be read correctly. This is because the backslash character is also used to indicate the start of an escape sequence, and the escape sequence **\"** stands for the double-quotation mark character. If you want to escape this sequence of characters, use a preceding backslash, like this: **\\"**. To summarize: If you need to write a file path that contains spaces or an end backslash, write it like this: **"C:\My Directory\ \"**.

Examples

Examples of the assignlicense command:

```
raptorxml assignlicense C:\licensepool\mylicensekey.altova_licenses
raptorxmlserver assignlicense C:\licensepool\mylicensekey.altova_licenses
raptorxml assignlicense --test-only=true C:\licensepool\mylicensekey.altova_licenses
```

- The first command above uploads the specified license to LicenseServer and assigns it to RaptorXML Server.
- The second command calls the server-executable to do the same thing as the first command.
- The last command uploads the specified license to LicenseServer and validates it, without assigning it to RaptorXML Server.

Options

Options are listed in short form (if available) and long form. You can use one or two dashes for both short and long forms. An option may or may not take a value. If it takes a value, it is written like this: **--option=value**. 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. If an option takes a Boolean value and no value is specified, then the option's default value is TRUE. Use the **--h, --help** option to display information about the command.

▼ test-only [t]

--t, --test-only = true|false
Values are true|false. If true, then the license file is uploaded to LicenseServer and validated, but not

 © 2017-2023 Altova GmbH

assigned.

5.9.3 verifylicense (Windows only)

Syntax and description

The **verifylicense** command checks whether the current product is licensed. Additionally, the `--license-key` option enables you to check whether a specific license key is already assigned to the product. This command is supported only on Windows systems. It is not supported on Linux or Mac systems.

```
raptorxml verifylicense [options]
raptorxmlserver verifylicense [options]
```

- To check whether a specific license is assigned to RaptorXML Server, supply the license key as the value of the `--license-key` option.
- The **verifylicense** command can be called from either executable: **raptorxml** or **raptorxmlserver**.

For details about licensing, see the LicenseServer documentation (https://www.altova.com/manual/en/licenseserver/3.12/).

▼ Casing and slashes on the command line

RaptorXML (and **RaptorXMLServer** for administration commands) *on Windows*
raptorxml (and **raptorxmlserver** for administration commands) *on Windows and Unix (Linux, Mac)*

* Note that lowercase (`raptorxml` and `raptorxmlserver`) works on all platforms (Windows, Linux, and Mac), while upper-lower (`RaptorXML`) works only on Windows and Mac.
* Use forward slashes on Linux and Mac, backslashes on Windows.

Examples

Example of the **verifylicense** command:

```
raptorxml verifylicense
raptorxml verifylicense --license-key=ABCD123-ABCD123-ABCD123-ABCD123-ABCD123-ABCD123
raptorxmlserver verifylicense --license-key=ABCD123-ABCD123-ABCD123-ABCD123-ABCD123-
ABCD123
```

- The first command checks whether RaptorXML Server is licensed.
- The second command checks whether RaptorXML Server is licensed with the license key specified with the `--license-key` option.
- The third command is the same as the second command, but is executed by the server-executable.

Options

Options are listed in short form (if available) and long form. You can use one or two dashes for both short and long forms. An option may or may not take a value. If it takes a value, it is written like this: **--option=value**. 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. If an option takes a Boolean value and no value is specified, then the option's default value is `TRUE`. Use the **`--h, --help`** option to display information about the command.

- license-key [l]

  `--l, --license-key =` *Value*

  Checks whether RaptorXML Server is licensed with the license key specified as the value of this option.

© 2017-2023 Altova GmbH

5.10 Administration Commands

Administration commands (such as installation-as-service and licensing commands) are issued to the server executable of RaptorXML Server (named **RaptorXMLServer**). This executable is located by default at:

Windows	`<ProgramFilesFolder>\Altova\RaptorXMLServer2024\bin\`**RaptorXMLServer.exe**
Linux	`/opt/Altova/RaptorXMLServer2024/bin/`**raptorxmlserver**
Mac	`/usr/local/Altova/RaptorXMLServer2024/bin/`**raptorxmlserver**

Usage

The command line syntax is:

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

- `--help` (short form `--h`) displays the help text of the given command. If no command is named, then all commands of the executable are listed, each with a brief description of the command.
- `--version` displays RaptorXML Server version number.
- `<command>` is the command to execute. Commands are displayed in the sub-sections of this section *(see list below)*.
- `[options]` are the options of a command; they are listed and described with their respective commands.
- `[arguments]` are the arguments of a command; they are listed and described with their respective commands.

▼ Casing and slashes on the command line

RaptorXML (and **RaptorXMLServer** for administration commands) *on Windows*
raptorxml (and **raptorxmlserver** for administration commands) *on Windows and Unix (Linux, Mac)*

* Note that lowercase (`raptorxml` and `raptorxmlserver`) works on all platforms (Windows, Linux, and Mac), while upper-lower (`RaptorXML`) works only on Windows and Mac.
* Use forward slashes on Linux and Mac, backslashes on Windows.

Administration commands

Commands of the server executable provide administration functionality. They are listed below and described in the sub-sections of this section:

- install 208
- uninstall 208
- start 209
- setdeflang 210
- licenseserver 211
- assignlicense 212
- verifylicense 214
- createconfig 215

- exportresourcestrings 216
- debug 217
- help 219

5.10.1 install

Syntax and description

The **install** command installs RaptorXML Server as a service on the server machine.

```
raptorxmlserver install [options]
```

- Note that installing RaptorXML Server as a service does not automatically start the service. To start the service, use the **start** command.
- To uninstall RaptorXML Server as a service, use the **uninstall** command.
- Use the **--h, --help** option to display information about the command.

▼ Casing and slashes on the command line

RaptorXML (and **RaptorXMLServer** for administration commands) *on Windows*
raptorxml (and **raptorxmlserver** for administration commands) *on Windows and Unix (Linux, Mac)*

* Note that lowercase (`raptorxml` and `raptorxmlserver`) works on all platforms (Windows, Linux, and Mac), while upper-lower (`RaptorXML`) works only on Windows and Mac.
* Use forward slashes on Linux and Mac, backslashes on Windows.

Example

Example of the **install** command:

```
raptorxmlserver install
```

5.10.2 uninstall

Syntax and description

The **uninstall** command uninstalls RaptorXML Server as a service on the server machine.

```
raptorxmlserver uninstall [options]
```

To re-install RaptorXML Server as a service, use the **install** command.

▼ Casing and slashes on the command line

RaptorXML (and **RaptorXMLServer** for administration commands) *on Windows*
raptorxml (and **raptorxmlserver** for administration commands) *on Windows and Unix (Linux, Mac)*

* Note that lowercase (`raptorxml` and `raptorxmlserver`) works on all platforms (Windows, Linux, and

 © 2017-2023 Altova GmbH

Mac), while upper-lower (RaptorXML) works only on Windows and Mac.
* Use forward slashes on Linux and Mac, backslashes on Windows.

Example

Example of the **uninstall** command:

```
raptorxmlserver uninstall
```

5.10.3 start

Syntax and description

The **start** command starts RaptorXML Server as a service on the server machine.

```
raptorxmlserver start [options]
```

- If RaptorXML Server is not installed as a service, install it first with the **install** command (before starting it).
- To uninstall RaptorXML Server as a service, use the **uninstall** command.
- Use the **--h, --help** option to display information about the command.

▼ Casing and slashes on the command line

RaptorXML (and **RaptorXMLServer** for administration commands) *on Windows*
raptorxml (and **raptorxmlserver** for administration commands) *on Windows and Unix (Linux, Mac)*

* Note that lowercase (raptorxml and raptorxmlserver) works on all platforms (Windows, Linux, and Mac), while upper-lower (RaptorXML) works only on Windows and Mac.
* Use forward slashes on Linux and Mac, backslashes on Windows.

▼ Backslashes, spaces, and special characters on Windows systems

On Windows systems: When spaces or special characters occur in strings (for example in file or folder names, or company, person or product names), use quotes: for example, **"My File"**. Note, however, that a backslash followed by a double-quotation mark (for example, **"C:\My directory\"**) might not be read correctly. This is because the backslash character is also used to indicate the start of an escape sequence, and the escape sequence **\"** stands for the double-quotation mark character. If you want to escape this sequence of characters, use a preceding backslash, like this: **\\"**. To summarize: If you need to write a file path that contains spaces or an end backslash, write it like this: **"C:\My Directory\ \"**.

Example

Example of the **start** command:

```
raptorxmlserver start
```

Options

Options are listed in short form (if available) and long form. You can use one or two dashes for both short and long forms. An option may or may not take a value. If it takes a value, it is written like this: **--option=value**. 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. If an option takes a Boolean value and no value is specified, then the option's default value is `TRUE`. Use the **--h, --help** option to display information about the command.

- config [c]

  --c, --config = *File*
  Specifies the path to a configuration file.

- fork

  --fork = `true|false`
  Provides the ability to fork when using classic `init` on Unix servers. The default is `false`.

- port

  --port = *PortNumber*
  The port number of the debug instance of RaptorXML Server.

5.10.4 setdeflang

Syntax and description

The **setdeflang** command (short form is **sdl**) sets the default language of RaptorXML Server. Available languages are English (`en`), German (`de`), Spanish (`es`), French (`fr`), and Japanese (`ja`). The command takes a mandatory *LanguageCode* argument.

raptorxml setdeflang [options] *LanguageCode*
raptorxmlserver setdeflang [options] *LanguageCode*

- The *LanguageCode* argument is required and sets the default language of RaptorXML Server. The respective values to use are: `en`, `de`, `es`, `fr`, `ja`.
- The **setdeflang** command can be called from either executable: **raptorxml** or **raptorxmlserver**.
- Use the **--h, --help** option to display information about the command.

- Casing and slashes on the command line

  RaptorXML (and **RaptorXMLServer** for administration commands) *on Windows*
  raptorxml (and **raptorxmlserver** for administration commands) *on Windows and Unix (Linux, Mac)*

  * Note that lowercase (`raptorxml` and `raptorxmlserver`) works on all platforms (Windows, Linux, and Mac), while upper-lower (`RaptorXML`) works only on Windows and Mac.
  * Use forward slashes on Linux and Mac, backslashes on Windows.

 © 2017-2023 Altova GmbH

Examples

Examples of the **setdeflang** (**sdl**) command:

```
raptorxml sdl de
raptorxml setdeflang es
raptorxmlserver setdeflang es
```

- The first command sets the default language of RaptorXML Server to German.
- The second command sets the default language of RaptorXML Server to Spanish.
- The third command is the same as the second command, but is executed by the server-executable.

Options

Use the **--h, --help** option to display information about the command.

5.10.5 licenseserver

Syntax and description

The **licenseserver** command registers RaptorXML Server with the Altova LicenseServer specified by the *Server-Or-IP-Address* argument. For the licenseserver command to be executed successfully, the two servers (RaptorXML Server and LicenseServer) must be on the same network and LicenseServer must be running. You must also have administrator privileges in order to register RaptorXML Server with LicenseServer.

```
raptorxml licenseserver [options] Server-Or-IP-Address
raptorxmlserver licenseserver [options] Server-Or-IP-Address
```

- The *Server-Or-IP-Address* argument takes the name or IP address of the LicenseServer machine.
- The **licenseserver** command can be called from either executable: **raptorxml** or **raptorxmlserver**.

Once RaptorXML Server has been successfully registered with LicenseServer, you will receive a message to this effect. The message will also display the URL of the LicenseServer. You can now go to LicenseServer to assign RaptorXML Server a license. For details about licensing, see the LicenseServer documentation (https://www.altova.com/manual/en/licenseserver/3.12/).

▼ Casing and slashes on the command line

RaptorXML (and **RaptorXMLServer** for administration commands) *on Windows*
raptorxml (and **raptorxmlserver** for administration commands) *on Windows and Unix (Linux, Mac)*

* Note that lowercase (raptorxml and raptorxmlserver) works on all platforms (Windows, Linux, and Mac), while upper-lower (RaptorXML) works only on Windows and Mac.
* Use forward slashes on Linux and Mac, backslashes on Windows.

▼ Backslashes, spaces, and special characters on Windows systems

On Windows systems: When spaces or special characters occur in strings (for example in file or folder names, or company, person or product names), use quotes: for example, **"My File"**. Note, however, that

a backslash followed by a double-quotation mark (for example, **"C:\My directory\"**) might not be read correctly. This is because the backslash character is also used to indicate the start of an escape sequence, and the escape sequence **\"** stands for the double-quotation mark character. If you want to escape this sequence of characters, use a preceding backslash, like this: **\\"**. To summarize: If you need to write a file path that contains spaces or an end backslash, write it like this: **"C:\My Directory\ \"**.

Examples

Examples of the **licenseserver** command:

```
raptorxml licenseserver DOC.altova.com
raptorxml licenseserver localhost
raptorxml licenseserver 127.0.0.1
raptorxmlserver licenseserver 127.0.0.1
```

The commands above specify, respectively, the machine named `DOC.altova.com`, and the user's machine (`localhost` and `127.0.0.1`) as the machine running Altova LicenseServer. In each case, the command registers RaptorXML Server with the LicenseServer on the machine specified. The last command calls the server-executable to execute the command.

Options

Options are listed in short form (if available) and long form. You can use one or two dashes for both short and long forms. An option may or may not take a value. If it takes a value, it is written like this: **--option=value**. 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. If an option takes a Boolean value and no value is specified, then the option's default value is `TRUE`. Use the **--h, --help** option to display information about the command.

- json [j]

  --j, --json = `true|false`

  Values are `true|false`. If `true`, prints the result of the registration attempt as a machine-parsable JSON object.

5.10.6 assignlicense (Windows only)

Syntax and description

The **assignlicense** command is available on Windows only (not on Linux or Mac systems). It uploads a license file to the Altova LicenseServer with which RaptorXML Server is registered (see the `licenseserver` command), and assigns the license to RaptorXML Server. It takes the path of a license file as its argument. The command also allows you to test the validity of a license.

```
raptorxml assignlicense [options] FILE
raptorxmlserver assignlicense [options] FILE
```

 © 2017-2023 Altova GmbH

- The *FILE* argument takes the path of the license file.
- The `--test-only` option uploads the license file to LicenseServer and validates the license, but does not assign the license to RaptorXML Server.
- The **assignlicense** command can be called from either executable: **raptorxml** or **raptorxmlserver**.

For details about licensing, see the LicenseServer documentation (https://www.altova.com/manual/en/licenseserver/3.12/).

▼ Casing and slashes on the command line

RaptorXML (and **RaptorXMLServer** for administration commands) *on Windows*
raptorxml (and **raptorxmlserver** for administration commands) *on Windows and Unix (Linux, Mac)*

* Note that lowercase (`raptorxml` and `raptorxmlserver`) works on all platforms (Windows, Linux, and Mac), while upper-lower (`RaptorXML`) works only on Windows and Mac.
* Use forward slashes on Linux and Mac, backslashes on Windows.

▼ Backslashes, spaces, and special characters on Windows systems

On Windows systems: When spaces or special characters occur in strings (for example in file or folder names, or company, person or product names), use quotes: for example, **"My File"**. Note, however, that a backslash followed by a double-quotation mark (for example, **"C:\My directory\"**) might not be read correctly. This is because the backslash character is also used to indicate the start of an escape sequence, and the escape sequence **\"** stands for the double-quotation mark character. If you want to escape this sequence of characters, use a preceding backslash, like this: **\\"**. To summarize: If you need to write a file path that contains spaces or an end backslash, write it like this: **"C:\My Directory\ \"**.

Examples

Examples of the `assignlicense` command:

```
raptorxml assignlicense C:\licensepool\mylicensekey.altova_licenses
raptorxmlserver assignlicense C:\licensepool\mylicensekey.altova_licenses
raptorxml assignlicense --test-only=true C:\licensepool\mylicensekey.altova_licenses
```

- The first command above uploads the specified license to LicenseServer and assigns it to RaptorXML Server.
- The second command calls the server-executable to do the same thing as the first command.
- The last command uploads the specified license to LicenseServer and validates it, without assigning it to RaptorXML Server.

Options

Options are listed in short form (if available) and long form. You can use one or two dashes for both short and long forms. An option may or may not take a value. If it takes a value, it is written like this: **--option=value**. 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. If an option takes a Boolean value and no value is specified, then the option's default value is `TRUE`. Use the **--h, --help** option to display information about the command.

- test-only [t]

  --t, --test-only = `true|false`
  Values are `true|false`. If `true`, then the license file is uploaded to LicenseServer and validated, but not assigned.

5.10.7 verifylicense (Windows only)

Syntax and description

The **verifylicense** command checks whether the current product is licensed. Additionally, the `--license-key` option enables you to check whether a specific license key is already assigned to the product. This command is supported only on Windows systems. It is not supported on Linux or Mac systems.

```
raptorxml verifylicense [options]
raptorxmlserver verifylicense [options]
```

- To check whether a specific license is assigned to RaptorXML Server, supply the license key as the value of the `--license-key` option.
- The **verifylicense** command can be called from either executable: **raptorxml** or **raptorxmlserver**.

For details about licensing, see the LicenseServer documentation (https://www.altova.com/manual/en/licenseserver/3.12/).

- Casing and slashes on the command line

  RaptorXML (and **RaptorXMLServer** for administration commands) *on Windows*
  raptorxml (and **raptorxmlserver** for administration commands) *on Windows and Unix (Linux, Mac)*

  * Note that lowercase (`raptorxml` and `raptorxmlserver`) works on all platforms (Windows, Linux, and Mac), while upper-lower (`RaptorXML`) works only on Windows and Mac.
  * Use forward slashes on Linux and Mac, backslashes on Windows.

Examples

Example of the **verifylicense** command:

```
raptorxml verifylicense
raptorxml verifylicense --license-key=ABCD123-ABCD123-ABCD123-ABCD123-ABCD123-ABCD123
raptorxmlserver verifylicense --license-key=ABCD123-ABCD123-ABCD123-ABCD123-ABCD123-
ABCD123
```

- The first command checks whether RaptorXML Server is licensed.
- The second command checks whether RaptorXML Server is licensed with the license key specified with the `--license-key` option.
- The third command is the same as the second command, but is executed by the server-executable.

 © 2017-2023 Altova GmbH

Options

Options are listed in short form (if available) and long form. You can use one or two dashes for both short and long forms. An option may or may not take a value. If it takes a value, it is written like this: **--option=value**. 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. If an option takes a Boolean value and no value is specified, then the option's default value is `TRUE`. Use the **--h, --help** option to display information about the command.

- license-key [l]

  --l, --license-key = *Value*
  Checks whether RaptorXML Server is licensed with the license key specified as the value of this option.

5.10.8 createconfig

Syntax and description

The **createconfig** command overwrites the server configuration file with default values.

```
raptorxmlserver createconfig [options]
```

- The **--lang** option specifies the default language of the server configuration file.

For more information about server configuration files, see Configuring the Server 242.

- Casing and slashes on the command line

  RaptorXML (and **RaptorXMLServer** for administration commands) *on Windows*
  raptorxml (and **raptorxmlserver** for administration commands) *on Windows and Unix (Linux, Mac)*

  * Note that lowercase (`raptorxml` and `raptorxmlserver`) works on all platforms (Windows, Linux, and Mac), while upper-lower (`RaptorXML`) works only on Windows and Mac.
  * Use forward slashes on Linux and Mac, backslashes on Windows.

Examples

Examples of the **createconfig** command:

```
raptorxml createconfig
raptorxml createconfig --lang=de
```

Options

- lang

  --lang = `en|de|es|fr|ja`

Specifies the default language of the server configuration file. The following options are available: English (`en`), German (`de`), Spanish (`es`), French (`fr`), Japanese (`ja`). If the option is not specified, English is chosen as the default language.

Use the **`--h, --help`** option to display information about the command.

Options are listed in short form (if available) and long form. You can use one or two dashes for both short and long forms. An option may or may not take a value. If it takes a value, it is written like this: **`--option=value`**. 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. If an option takes a Boolean value and no value is specified, then the option's default value is `TRUE`. Use the **`--h, --help`** option to display information about the command.

5.10.9 exportresourcestrings

Syntax and description

The **`exportresourcestrings`** command outputs an XML file containing the resource strings of the RaptorXML Server application in the specified language. Available export languages are English (`en`), German (`de`), Spanish (`es`), French (`fr`), and Japanese (`ja`).

```
raptorxml exportresourcestrings [options] LanguageCode XMLOutputFile
raptorxmlserver exportresourcestrings [options] LanguageCode XMLOutputFile
```

- The ***`LanguageCode`*** argument gives the language of the resource strings in the output XML file; this is the *export language*. Allowed export languages (with their language codes in parentheses) are: English (`en`), German, (`de`), Spanish (`es`), French (`fr`), and Japanese (`ja`).
- The ***`XMLOutputFile`*** argument specifies the path and name of the output XML file.
- The **`exportresourcestrings`** command can be called from either executable: **`raptorxml`** or **`raptorxmlserver`**.

How to create localizations is described below.

▼ Casing and slashes on the command line

`RaptorXML` (and **`RaptorXMLServer`** for administration commands) *on Windows*
`raptorxml` (and **`raptorxmlserver`** for administration commands) *on Windows and Unix (Linux, Mac)*

* Note that lowercase (`raptorxml` and `raptorxmlserver`) works on all platforms (Windows, Linux, and Mac), while upper-lower (`RaptorXML`) works only on Windows and Mac.
* Use forward slashes on Linux and Mac, backslashes on Windows.

▼ Backslashes, spaces, and special characters on Windows systems

On Windows systems: When spaces or special characters occur in strings (for example in file or folder names, or company, person or product names), use quotes: for example, **`"My File"`**. Note, however, that a backslash followed by a double-quotation mark (for example, **`"C:\My directory\"`**) might not be read correctly. This is because the backslash character is also used to indicate the start of an escape

 © 2017-2023 Altova GmbH

sequence, and the escape sequence **\"** stands for the double-quotation mark character. If you want to escape this sequence of characters, use a preceding backslash, like this: **\\"**. To summarize: If you need to write a file path that contains spaces or an end backslash, write it like this: **"C:\My Directory\ \"**.

Examples

Examples of the **exportresourcestrings** command:

```
raptorxml exportresourcestrings de c:\Strings.xml
raptorxmlserver exportresourcestrings de c:\Strings.xml
```

- The first command above creates a file called `Strings.xml` at `c:\` that contains the resource strings of RaptorXML Server in German.
- The second command calls the server-executable to do the same thing as the first example.

Creating localized versions of RaptorXML Server

You can create a localized version of RaptorXML Server for any language of your choice. Five localized versions (English, German, Spanish, French, and Japanese) are already available in the `C:\Program Files (x86)\Altova\RaptorXMLServer2024\bin` folder, and therefore do not need to be created.

Create a localized version as follows:

1. Generate an XML file containing the resource strings by using the `exportresourcestrings` command (*see command syntax above*). The resource strings in this XML file will be one of the five supported languages: English (`en`), German (`de`), Spanish (`es`), French (`fr`), or Japanese (`ja`), according to the ***LanguageCode*** argument used with the command.
2. Translate the resource strings from one of the five supported languages 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 Server DLL file from your translated XML file.
4. After you receive your localized DLL file from Altova Support, save the DLL in the `C:\Program Files (x86)\Altova\RaptorXMLServer2024\bin` folder. Your DLL file will have a name of the form `RaptorXML2024_lc.dll`. The `_lc` part of the name contains the language code. For example, in `RaptorXML2024_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 Server application to use. For the argument of the `setdeflang` command, use the language code that is part of the DLL name.

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

5.10.10 debug

Syntax and description

The **debug** command starts RaptorXML Server for debugging—not as a service. To stop RaptorXML Server in this mode, press **Ctrl+C**.

```
raptorxmlserver debug [options]
```

- Casing and slashes on the command line

  RaptorXML (and **RaptorXMLServer** for administration commands) *on Windows*
  raptorxml (and **raptorxmlserver** for administration commands) *on Windows and Unix (Linux, Mac)*

  * Note that lowercase (raptorxml and raptorxmlserver) works on all platforms (Windows, Linux, and Mac), while upper-lower (RaptorXML) works only on Windows and Mac.
  * Use forward slashes on Linux and Mac, backslashes on Windows.

- Backslashes, spaces, and special characters on Windows systems

  On Windows systems: When spaces or special characters occur in strings (for example in file or folder names, or company, person or product names), use quotes: for example, **"My File"**. Note, however, that a backslash followed by a double-quotation mark (for example, **"C:\My directory\"**) might not be read correctly. This is because the backslash character is also used to indicate the start of an escape sequence, and the escape sequence **\"** stands for the double-quotation mark character. If you want to escape this sequence of characters, use a preceding backslash, like this: **\\"**. To summarize: If you need to write a file path that contains spaces or an end backslash, write it like this: **"C:\My Directory\ \"**.

Example

Example of the **debug** command:

```
raptorxmlserver debug
```

Options

Options are listed in short form (if available) and long form. You can use one or two dashes for both short and long forms. An option may or may not take a value. If it takes a value, it is written like this: **--option=value**. 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. If an option takes a Boolean value and no value is specified, then the option's default value is TRUE. Use the **--h, --help** option to display information about the command.

- config [c]

  --c, --config = *File*
  Specifies the path to a configuration file.

- port

  --port = *PortNumber*
  The port number of the debug instance of RaptorXML Server

 © 2017-2023 Altova GmbH

5.10.11 help

Syntax and description

The **help** command takes a single argument (Command), which is the name of the command for which help is required. It displays the command's syntax, its options, and other relevant information. If the Command argument is not specified, then all commands of the executable are listed, with each having a brief text description. The **help** command can be called from either executable: **raptorxml** or **raptorxmlserver**.

```
raptorxml help Command
raptorxmlserver help Command
```

- Casing and slashes on the command line

  RaptorXML (and **RaptorXMLServer** for administration commands) *on Windows*
  raptorxml (and **raptorxmlserver** for administration commands) *on Windows and Unix (Linux, Mac)*

  * Note that lowercase (raptorxml and raptorxmlserver) works on all platforms (Windows, Linux, and Mac), while upper-lower (RaptorXML) works only on Windows and Mac.
  * Use forward slashes on Linux and Mac, backslashes on Windows.

Example

Examples of the **help** command to display information about the **licenserver** command (this command is available in both executables):

```
raptorxml help licenseserver
raptorxmlserver help licenseserver
```

The --help option

Help information about a command is also available by using the --help option of the command for which help information is required. The two commands below produce the same results:

```
raptorxml licenseserver --help
```

The command above uses the --help option of the **licenseserver** command.

```
raptorxml help licenseserver
```

The **help** command takes licenseserver as its argument.

Both commands display help information about the **licenseserver** command.

5.10.12 version

Syntax and description

The **version** command displays the version number of RaptorXML Server. It can be called from either executable: **raptorxml** or **raptorxmlserver**.

```
raptorxml version
raptorxmlserver version
```

▼ Casing and slashes on the command line

RaptorXML (and **RaptorXMLServer** for administration commands) *on Windows*
raptorxml (and **raptorxmlserver** for administration commands) *on Windows and Unix (Linux, Mac)*

* Note that lowercase (`raptorxml` and `raptorxmlserver`) works on all platforms (Windows, Linux, and Mac), while upper-lower (`RaptorXML`) works only on Windows and Mac.
* Use forward slashes on Linux and Mac, backslashes on Windows.

Example

Examples of the `version` command:

```
raptorxml version
raptorxmlserver version
```


© 2017-2023 Altova GmbH

5.11 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 221
- Messages, Errors, Help 222
- Processing 223
- XML 224
- XSD 225
- XQuery 227
- XSLT 229
- JSON/Avro 231
- XML Signatures 232

5.11.1 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\RaptorXMLServer2024\etc\RootCatalog.xml`). See the section, XML Catalogs 47, 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 47, for information about working with catalogs.

▼ enable-globalresources

`--enable-globalresources =` `true|false`
Enables global resources 53. 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 53 (and enables global resources) 53.

▼ globalresourcefile [gr]

`--gr | --globalresourcefile =` *`FILE`*
Specifies the global resource file 53 (and enables global resources) 53.

▼ recurse

`--recurse =` `true|false`
Used to select files within sub-directories, including in ZIP archives. 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. References to ZIP files must be given in quotes. The wildcard characters `*` and `?` may be used. So, `*.xml` will select all `.xml` files in the (zip) folder. The option's default value is `false`.
Note: Boolean option values are set to `true` if the option is specified without a value.

5.11.2 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 | unlimited`*
  Specifies the error limit with a value range of `1` to `9999` or `unlimited`. The default value is `100`. Processing stops when the error limit is reached. Useful for limiting processor use during validation/transformation.

- 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`.)

- info-limit

  `--info-limit =` *`N | unlimited`*
  Specifies the information message limit in the range `1-65535` or `unlimited`. Processing continues if the specified info limit is reached, but further messages are not reported. The default value is `100`.

- log-output

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

- network-timeout

  `--network-timeout =` *`VALUE`*
  Specifies the timeout in milliseconds for remote I/O operations. Default is: `40000`.

- 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

 © 2017-2023 Altova GmbH

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

▼ version

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

▼ warning-limit

--warning-limit = *N | unlimited*
Specifies the warning limit in the range `1-65535` or `unlimited`. Processing continues if this limit is reached, but further warnings are not reported. The default value is `100`.

5.11.3 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. Add the option multiple times to specify more than one script.

▼ script-api-version

--api, --script-api-version = `1; 2; 2.1 to 2.4; 2.4.1; 2.5 to 2.8; 2.8.1 to 2.8.6`
Specifies the Python API version to be used for the script. The default value is the latest version, currently **`2.8.6`**. Instead of integer values such as `1` and `2`, you can also use the corresponding values `1.0` and `2.0`. Similarly, you can use the three-digit `2.5.0` for the two-digit `2.5`. Also see the topic Python API Versions 353.

▼ script-param

--script-param = *KEY:VALUE*

Additional user-specified parameters that can be accessed during the execution of Python scripts. Add the option multiple times to specify more than one script parameter.

▼ 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.

▼ xml-validation-error-as-warning

--xml-validation-error-as-warning = `true|false`

If `true`, treats validation errors as warnings. If errors are treated as warnings, additional processing, such as XSLT transformations, will continue regardless of errors. Default is `false`.

5.11.4 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: `true`.

▼ 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.

 © 2017-2023 Altova GmbH

- 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 for the XML instance document: `wf`=wellformed check; `id`=wellformed with ID/IDREF checks; `valid`=validation. Default value is `wf`. Note that a value of `valid` requires that each instance document loaded during processing references a DTD. If no DTD exists, an error is reported.

- xml-validation-error-as-warning

  --xml-validation-error-as-warning = `true|false`
  If `true`, treats validation errors as warnings. If errors are treated as warnings, additional processing, such as XSLT transformations, will continue regardless of errors. Default is `false`.

- 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 more than one schema document.

5.11.5 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.

- ct-restrict-mode

  --ct-restrict-mode = `1.0|1.1|default`
  Specifies how to check complex type restrictions. A value of **`1.0`** checks complex type restrictions as defined in the XSD 1.0 specification—even in XSD 1.1 validation mode. A value of **`1.1`** checks complex type restrictions as defined in the XSD 1.1 specification—even in XSD 1.0 validation mode. A value of **`default`** checks complex type restrictions as defined in the XSD specification of the current validation mode (1.0 or 1.1). The default value is **`default`**.

- 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.

▼ report-import-namespace-mismatch-as-warning

--report-import-namespace-mismatch-as-warning = true|false
Downgrades namespace or target-namespace mismatch errors when importing schemas with **xs:import** from errors to warnings. The 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 option specifies whether to load a schema document or just license a namespace, and, if a schema document is to be loaded, which information should be used to find it. Default: load-preferring-schemalocation.
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[47]. 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[47]. If no schemaLocation attribute is present, then the value of the namespace attribute is used via a catalog mapping[47]. This is the **default value**.
- load-by-namespace: The value of the namespace attribute is used to locate the schema via a catalog mapping[47].
- load-combining-both: If either the namespace or schemaLocation attribute has a catalog mapping[47], then the mapping is used. If both have catalog mappings[47], then the value of the --schema-mapping option (XML/XSD option[225]) decides which mapping is used. If no catalog mapping[47] 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
Specifies the behavior of the xsi:schemaLocation and xsi:noNamespaceSchemaLocation attributes: Whether to load a schema document, and, if yes, which information should be used to find it. Default: load-by-schemalocation.

- The load-by-schemalocation value uses the URL of the schema location[383] in the xsi:schemaLocation and xsi:noNamespaceSchemaLocation attributes in XML instance documents. This is the **default value**.
- The load-by-namespace value takes the namespace part[383] of xsi:schemaLocation and an empty string in the case of xsi:noNamespaceSchemaLocation and locates the schema via a catalog mapping[47].
- If load-combining-both is used and if either the namespace part or the URL part has a catalog mapping[47], then the catalog mapping[47] is used. If both have catalog mappings[47], then the value of the --schema-mapping option (XML/XSD option[225]) 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

 © 2017-2023 Altova GmbH

--schema-mapping = `prefer-schemalocation | prefer-namespace`
If schema location and namespace are both used to find a schema document, specifies which of them should be preferred during catalog lookup. (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 [47], 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`.

- xml-mode-for-schemas

--xml-mode-for-schemas = `wf|id|valid`
Specifies the XML processing mode to use for XML schema documents: `wf`=wellformed check; `id`=wellformed with ID/IDREF checks; `valid`=validation. Default value is `wf`. Note that a value of `valid` requires that each schema document loaded during processing references a DTD. If no DTD exists, an error is reported.

- 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`.

5.11.6 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, xsltoutput

output = *FILE*, **xsltoutput =** *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. Additional output files, such as generated image files, are reported as `xslt-additional-output-files`. If no `--output` or `--xsltoutput` 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;
```

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:*

```
raptorxml xslt --input=c:\Test.xml --output=c:\Output.xml --
param=date://node[1]/@att1 --p=title:'stringwithoutspace' --param=title:"'string
```

 © 2017-2023 Altova GmbH

```
with spaces'" --p=amount:456 c:\Test.xslt
```

- updated-xml

  --updated-xml = discard|writeback|asmainresult
  Specifies how the updated XML file should be handled.

  - discard: The update is discarded and not written to file. Neither the input file nor the output file will be updated. Note that this is the default.
  - writeback: Writes the update back to the input XML file that is specified with the --input option.
  - asmainresult: Writes the update to the output XML file that is specified with the --output option. If the **--output** option is not specified, then the update is written to the standard output. In both cases, the input XML file will not be modified.

  Default is discard.

- xpath-static-type-errors-as-warnings

  --xpath-static-type-errors-as-warnings = true|false
  If true, downgrades to warnings any type errors that are detected in the XPath static context. Whereas an error would cause the execution to fail, a warning would enable processing to continue. Default is false.

- xquery-update-version

  --xquery-update-version = 1|1.0|3|3.0|
  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|1.0|3|3.0|3.1
  Specifies whether the XQuery processor should use XQuery 1.0 or XQuery 3.0. Default value is 3.1.

5.11.7 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 path to the folder that contains the barcode extension file `AltovaBarcodeExtension.jar`. The path must be given in one of the following forms:

- A file URI, for example: `--javaext-barcode-location="file:///C:/Program Files/Altova/RaptorXMLServer2024/etc/jar/"`
- A Windows path with backslashes escaped, for example: `--javaext-barcode-location="C:\\Program Files\\Altova\\RaptorXMLServer2024\\etc\\jar\\"`

▼ 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, xsltoutput

output = *FILE*, **xsltoutput =** *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. Additional output files, such as generated image files, are reported as `xslt-additional-output-files`. If no `--output` or `--xsltoutput` 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;
```

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

 © 2017-2023 Altova GmbH

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:*

```
raptorxml 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.

- initial-template, template-entry-point

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

- initial-mode, template-mode

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

- xpath-static-type-errors-as-warnings

  `--xpath-static-type-errors-as-warnings =` `true|false`
  If `true`, downgrades to warnings any type errors that are detected in the XPath static context. Whereas an error would cause the execution to fail, a warning would enable processing to continue. Default is `false`.

- xslt-version

  `--xslt-version =` `1|1.0|2|2.0|3|3.0|3.1`
  Specifies whether the XSLT processor should use XSLT 1.0, XSLT 2.0, or XSLT 3.0. Default value is `3`.

5.11.8 JSON/Avro

- additional-schema

  `--additional-schema =` *FILE*
  Specifies URIs of an additional schema document. The additional schema will be loaded by the main schema and can be referenced from the main schema by the additional schemas **`id`** or **`$id`** property.

- disable-format-checks

--disable-format-checks = `true|false`
Disables the semantic validation imposed by the format attribute. Default value is `false`.
Note: Boolean option values are set to `true` if the option is specified without a value.

- jsonc

--jsonc = `true|false`
Enables support for comments in JSON. Default value is `false`.
Note: Boolean option values are set to `true` if the option is specified without a value.

- json-lines

--json-lines = `true|false`
Enables support for JSON Lines (that is, one JSON value per line). Default value is `false`.
Note: Boolean option values are set to `true` if the option is specified without a value.

5.11.9 XML Signatures

- absolute-reference-uri

--absolute-reference-uri = `true|false`
Specifies whether the URI of the signed document is to be read as absolute (`true`) or relative (`false`). Default is `false`.
Note: Boolean option values are set to `true` if the option is specified without a value.

- certname, certificate-name

--certname, --certificate-name = *VALUE*
The name of the certificate used for signing.

Windows
This is the **Subject** name of a certificate from the selected **--certificate-store**.

Example to list the certificates (under PowerShell)

```
% ls cert://CurrentUser/My
PSParentPath: Microsoft.PowerShell.Security\Certificate::CurrentUser\My
Thumbprint Subject
---------- -------
C9DF64BB0AAF5FA73474D78B7CCFFC37C95BFC6C CN=certificate1
... CN=...
```

Example: **--certificate-name==certificate1**

Linux/MacOS
--certname specifies the file name of a PEM encoded X.509v3 certificate with the private key. Such files usually have the extension **.pem**.

xample: **--certificate-name==/path/to/certificate1.pem**

 © 2017-2023 Altova GmbH

- certstore, certificate-store

  --certstore, --certificate-store = *VALUE*
  The location where the certificate specified with **--certificate-name** is stored.

  Windows
  The name of a certificate store under **cert://CurrentUser**. The available certificate stores can be listed (under PowerShell) by using **% ls cert://CurrentUser/**. Certificates would then be listed as follows:

```
Name : TrustedPublisher
Name : ClientAuthIssuer
Name : Root
Name : UserDS
Name : CA
Name : ACRS
Name : REQUEST
Name : AuthRoot
Name : MSIEHistoryJournal
Name : TrustedPeople
Name : MyCertStore
Name : Local NonRemovable Certificates
Name : SmartCardRoot
Name : Trust
Name : Disallowed
```

  Example: **--certificate-store==MyCertStore**

  Linux/MacOS
  The **--certstore** option is currently not supported.

- digest, digest-method

  --digest, --digest-method = sha1|sha256|sha384|sha512
  The algorithm that is used to compute the digest value over the input XML file. Available values are: sha1|sha256|sha384|sha512.

- hmackey, hmac-secret-key

  --hmackey, --hmac-secret-key = *VALUE*
  The HMAC shared secret key; must have a minimum length of six characters.

  Example: **--hmackey=secretpassword**

- hmaclen, hmac-output-length

  --hmaclen, --hmac-output-length = *LENGTH*
  Truncates the output of the HMAC algorithm to length bits. If specified, this value must be
  - a multiple of 8
  - larger than 80
  - larger than half of the underlying hash algorithm's output length

- keyinfo, append-keyinfo

  --keyinfo, --append-keyinfo = true|false

Specifies whether to include the `KeyInfo` element in the signature or not. The default is `false`.

▼ sigc14nmeth, signature-canonicalization-method

`--sigc14nmeth, --signature-canonicalization-method =` *VALUE*
Specifies the canonicalization algorithm to apply to the **`SignedInfo`** element. The value must be one of:

- `REC-xml-c14n-20010315`
- `xml-c14n11`
- `xml-exc-c14n#`

▼ sigmeth, signature-method

`--sigmeth, --signature-method =` *VALUE*
Specifies the algorithm to use for generating the signature.

When a certificate is used
If a certificate is specified, then `SignatureMethod` is optional and the value for this parameter is derived from the certificate. If specified, it must match the algorithm used by the certificate. Example: `rsa-sha256`.

When --hmac-secret-key is used
When `HMACSecretKey` is used, then `SignatureMethod` is mandatory. The value must be one of the supported HMAC algorithms:

- `hmac-sha256`
- `hmac-sha386`
- `hmac-sha512`
- `hmac-sha1` (discouraged by the specification)

Example: **`hmac-sha256`**

▼ sigtype, signature-type

`--sigtype, --signature-type =` `detached | enveloping | enveloped`
Specifies the type of signature to be generated.

▼ transforms

`--transforms =` *VALUE*
Specifies the XML Signature transformations applied to the input document. The supported values are:

- `REC-xml-c14n-20010315` for Canonical XML 1.0 (omit comments)
- `xml-c14n11` for Canonical XML 1.1 (omit comments)
- `xml-exc-c14n#` for Exclusive XML Canonicalization 1.0 (omit comments)
- `REC-xml-c14n-20010315#WithComments` for Canonical XML 1.0 (with comments)
- `xml-c14n11#WithComments` for Canonical XML 1.1 (with comments)
- `xml-exc-c14n#WithComments` for Exclusive XML Canonicalization 1.0 (with comments)
- `base64`
- `strip-whitespaces` Altova extension

Example: **`--transforms=xml-c14n11`**

© 2017-2023 Altova GmbH

Note: This option can be specified multiple times. If specified multiple times, then the order of specification is significant. The first specified transformation receives the input document. The last specified transformation is used immediately before calculation of the digest value.

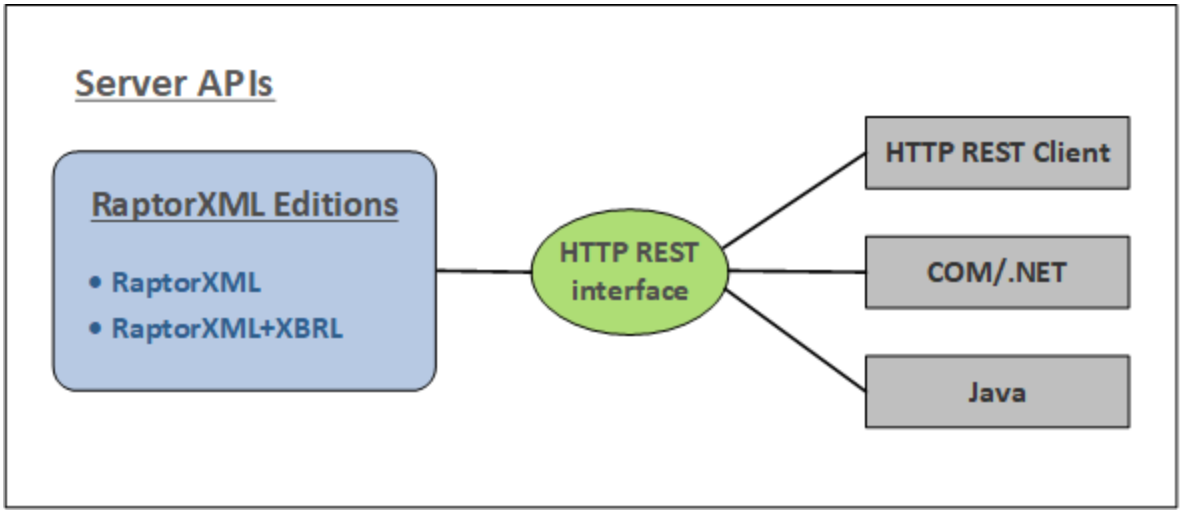
▼ write-default-attributes

--write-default-attributes = `true|false`

Specifies whether to include default attribute values from the DTD in the signed document.

6 Server APIs: HTTP REST, COM/.NET, Java

RaptorXML Server defines an HTTP REST interface, which is used by clients to dispatch jobs to the server. Clients can either access the HTTP REST interface directly or use the high-level COM/.NET and Java Server APIs. These APIs provide easy-to-use COM/.NET and Java classes which manage the creation and dispatch of the HTTP REST requests. The figure below shows a summary of the available HTTP REST client methods to communicate with the RaptorXML server.



There are three server APIs that can be used to communicate with RaptorXML via the HTTP REST interface (*also see figure above*).

- HTTP REST client interface 238
- COM/.NET API 274
- Java API 283

Note: The server APIs offer similar functionality as the command line interface (CLI) 56. This includes validation and document transformations. If you wish to use advanced functionality, such as reading, extracting, and analysing data, then use the Engine APIs. The Engine APIs can provide additional information such as the count of elements, their positions in the document, and complex XBRL data access and manipulation.

Usage

RaptorXML Server should be installed on a machine that is accessible by clients over the local network. Once the RaptorXML Server service has been started, clients can connect to the server and issue commands. The following access methods are labeled as Server APIs because they provide a way to communicate with a remote RaptorXML server.

- *HTTP REST client interface* 238: Client requests are made in JSON format as described in the section HTTP REST Client Interface 238. Each request is assigned a job directory on the server, in which output files are saved. The server responds to the client with all the information relevant to the job.
- *COM/.NET API* 274 *and Java API* 283: Applications and scripts in COM/.NET programming languages 274 and Java 283 applications use objects of the RaptorXML Server API 286 to access

 © 2017-2023 Altova GmbH

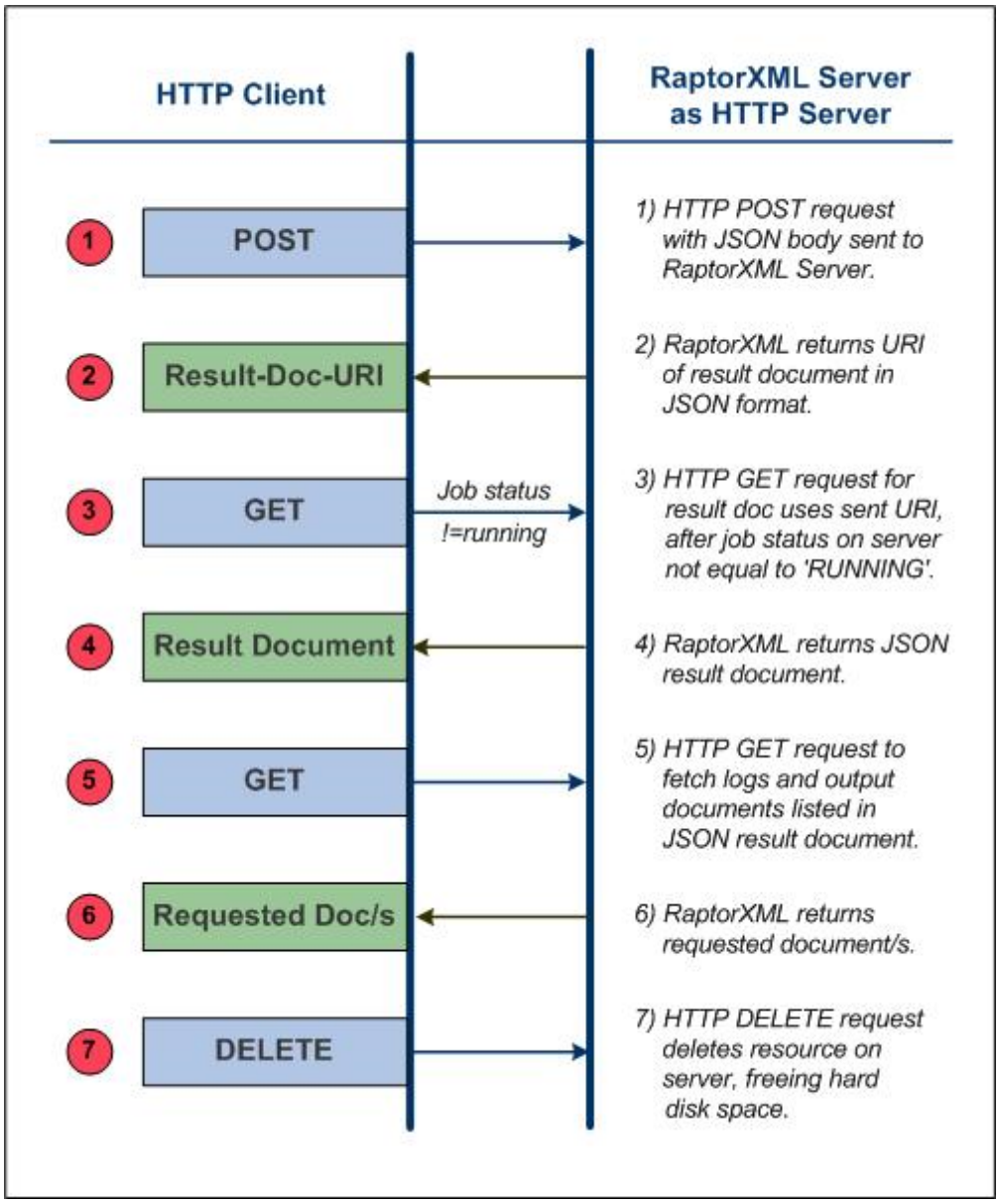
functionality of RaptorXML Server. The RaptorXML Server API[286] will issue the corresponding HTTP REST requests on behalf of the client. See the respective sub-sections for more information.

Licensing

RaptorXML Server is licensed on the machine on which it is installed. Connections to RaptorXML Server are made via HTTP.

6.1 HTTP REST Client Interface

RaptorXML Server accepts jobs submitted via HTTP (or HTTPS[247]). 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 REST interface

The HTTP REST interface, by default, allows result documents to be written to any location specified by the client (that is accessible with the HTTP protocol).

© 2017-2023 Altova GmbH

It is important therefore to consider this security aspect when configuring RaptorXML 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`[243] 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 Server must be started and properly configured. How to do this is described in the section Server Setup[239]. How to send client requests is described in the section Client Requests[250]. Finally, the section C# Example for REST API[270] provides a description of the REST API example file that is installed with your RaptorXML Server package.

6.1.1 Server Setup

RaptorXML must be licensed on the machine on which it is installed. This installation can then be accessed via an HTTP REST Interface[238]. To correctly set up RaptorXML Server, do the following. We assume that RaptorXML Server has already been correctly installed[46] and licensed[46].

1. RaptorXML Server must be either started as a service or an application[240] in order for it to be correctly accessed via HTTP or HTTPS. How to do this differs according to operating system and is described here: on Windows[240], on Linux[240], on macOS[241].
2. Use the initial server configuration[242] to test the connection to the server[241]. (The initial server configuration[242] 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[242], `server_config.xml`. If you wish to change any settings[243] in the file, edit the server configuration file and save the changes. HTTPS is disabled by default, and will need to be enabled in the configuration file[242].
4. If you have edited the server configuration file[242], then restart RaptorXML 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 that is used, and license errors are reported in the system log. So, refer to the system log[243] if there are problems with the server.

For more information about HTTPS, see the section HTTPS Settings[247].

6.1.1.1 Starting the Server

This section:

- Location of the Server executable 240
- Starting RaptorXML as a service on Windows 240
- Starting RaptorXML as a service on Linux 240
- Starting RaptorXML as a service on macOS 241

Location of the Server executable file

The RaptorXML Server executable is installed by default in the folder:

```
<ProgramFilesFolder>\Altova\RaptorXMLServer2024\bin\RaptorXML.exe
```

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

Starting as a service on Windows

The installation process will have registered RaptorXML Server as a service on Windows. You must, however, **start** RaptorXML 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 "AltovaRaptorXMLServer"`**
- Via the RaptorXML Server executable in a command prompt window: **`RaptorXMLServer.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` 243 setting of the server configuration file 242. 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 Server as a service with the following command:

[< Debian 8]	**`sudo /etc/init.d/raptorxmlserver start`**
[≥ Debian 8]	**`sudo systemctl start raptorxmlserver`**
[< CentOS 7]	**`sudo initctl start raptorxmlserver`**
[≥ CentOS 7]	**`sudo systemctl start raptorxmlserver`**
[< Ubuntu 15]	**`sudo initctl start raptorxmlserver`**
[≥ Ubuntu 15]	**`sudo systemctl start raptorxmlserver`**
[RedHat]	**`sudo initctl start raptorxmlserver`**

© 2017-2023 Altova GmbH

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

[< Debian 8]	**sudo /etc/init.d/raptorxmlserver stop**
[≥ Debian 8]	**sudo systemctl stop raptorxmlserver**
[< CentOS 7]	**sudo initctl stop raptorxmlserver**
[≥ CentOS 7]	**sudo systemctl stop raptorxmlserver**
[< Ubuntu 15]	**sudo initctl stop raptorxmlserver**
[≥ Ubuntu 15]	**sudo systemctl stop raptorxmlserver**
[RedHat]	**sudo initctl stop raptorxmlserver**

Starting as a service on macOS

Start RaptorXML Server as a service with the following command:

```
sudo launchctl load /Library/LaunchDaemons/com.altova.RaptorXMLServer2024.plist
```

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

```
sudo launchctl unload /Library/LaunchDaemons/com.altova.RaptorXMLServer2024.plist
```

6.1.1.2 Testing the Connection

This section:

- GET request to test the connection 242
- Server response and JSON data structure listing 242

GET request to test the connection

After RaptorXML 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 Server is specified in the server configuration file, `server_config.xml`, which is described in the next section, Server Configuration 242.

Server response and JSON data structure listing

If the service is running and the server is correctly configured, the request should never fail. RaptorXML 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[242]—you should test the connection again.

6.1.1.3 Configuring the Server

This section:

- Server configuration file: initial settings[242]
- Server configuration file: modifying the initial settings, reverting to initial settings[242]
- Server configuration file: listing and settings[243]
- Server configuration file: description of settings[243]
- Configuring the server address[246]

Server configuration file: initial settings

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

```
C:\Program Files (x86)\Altova\RaptorXMLServer2024\etc\server_config.xml
```

The initial configuration for RaptorXML 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\RaptorXMLServer2024\Output\`.

Other default settings are shown in the listing[243] 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[243]), save it, and then restart RaptorXML 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`:

```
RaptorXMLServer.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.

 © 2017-2023 Altova GmbH

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\RaptorXMLServer2024\output\</server.output-
root-dir>
  <server.script-root-dir>C:\Program
Files\Altova\RaptorXMLServer2024\etc\scripts\</server.script-root-dir>
  <!--<server.default-script-api-version>2</server.default-script-api-version>-->
    <!--<server.catalog-file>catalog.xml</server.catalog-file>-->
  <!--<server.log-file>C:
\ProgramData\Altova\RaptorXMLServer2024\Log\server.log</server.log-file>-->

  <http.enable>true</http.enable>
    <http.environment>production</http.environment>
    <http.socket-host>127.0.0.1</http.socket-host>
    <http.socket-port>8087</http.socket-port>
  <http.log-screen>true</http.log-screen>
    <http.access-file>C:\ProgramData\Altova\RaptorXMLServer2024\Log\access.log</http.access-
    file>
    <http.error-file>C:\ProgramData\Altova\RaptorXMLServer2024\Log\error.log</http.error-
    file>

  <https.enable>false</https.enable>
  <https.socket-host>127.0.0.1</https.socket-host>
  <https.socket-port>443</https.socket-port>
  <https.private-key>C:\Program
Files\Altova\RaptorXMLServer2024\etc\cert\key.pem</https.private-key>
  <https.certificate>C:\Program
Files\Altova\RaptorXMLServer2024\etc\cert\cert.pem</https.certificate>
  <!--<https.certificate-chain>/path/to/chain.pem</https.certificate-chain>-->

</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|fr|ja` (English, German, Spanish, French, and Japanese, respectively). See Localization Commands 199 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). Note, however, that local file paths cannot be used to access files from a remote machine via HTTP. So, if RaptorXML Server is running on a remote machine, set the value of this option to **`false`**. Setting the value to **`true`** is only viable if the client and server are on the same machine and you want to write the output files to a directory on that machine.
- When set to `false`, files will be written to the job's directory in the output directory[243], and the URIs of these files will be included in the result document[264]. 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[352] 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'*[352].

server.default-script-api-version
Default Python API version used to run Python scripts. By default the newest version of the Python API is used. Currently supported values are 1 and 2.

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\RaptorXMLServer2024\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.enable
A boolean value to enable or disable HTTP: `true | false`. HTTP can be enabled/disabled independently of HTTPS, and both can be active concurrently.

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.

 © 2017-2023 Altova GmbH

http.socket-host
The interface via which RaptorXML Server is accessed. If you wish RaptorXML 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 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 Server is started with the command `RaptorXMLServer.exe debug`, (see Starting the Server 240) 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.

http.max_request_body_size
This option specifies the maximum size, in bytes, of the request body that RaptorXML Server accepts. The default value is 100 MB. If the size of a request body is larger than the value specified for this option, then the server responds with `HTTP Error 413: Request entity too large`. The option's value must be greater than or equal to zero. The limit can be disabled by setting `http.max_request_body_size=0`.

https.enable
A boolean value to enable or disable HTTPS: `true | false`. HTTPS can be enabled/disabled independently of HTTP, and both can be active concurrently. HTTPS support is disabled by default and must be enabled by changing the value of this setting to **`true`**.

https.socket-host
Takes a string value which is the host address on which HTTPS connections are accepted. To accept connections from the local host only, set **`localhost`** or **`127.0.0.1`**. If you wish RaptorXML Server to accept connections from all remote machines, set the value to: `0.0.0.0`, like this: `<https.socket-host>0.0.0.0</https.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 Server: port `8087`; Altova FlowForce Server: port `8082`. You can also use IPv6 addresses such as: `'::'`.

`https.socket-port`
An integer value that is the port on which HTTPS is accepted. The port must be fixed and known so that HTTP requests can be correctly addressed to the service.

`https.private-key, https.certificate`
URIs that are the paths, respectively, to the server's private key and certificate files. Both are required. See HTTPS Settings 247 and Setting Up SSL Encryption 247 for more information. On Windows machines, you can also use Windows paths.

`https.certificate-chain`
An optional setting, this is a URI which locates the intermediate certificate file. If you have two intermediate certificates (primary and secondary), then combine them into one file as described in Step 7 at Setting Up SSL Encryption 247. See HTTPS Settings 247 and Setting Up SSL Encryption 247 for more information.

The RaptorXML 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 `123.12.123.1`, 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 Server can then be addressed with:

```
http://123.12.123.1:8087/
```

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

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

Note: Messages submitted to RaptorXML 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).

 © 2017-2023 Altova GmbH

6.1.1.4 HTTPS Settings

RaptorXML Server supports startup not only as an HTTP server, but also as an HTTPS server. Both types of connection may be active concurrently.

Enabling HTTPS

HTTPS support is disabled by default. To enable HTTPS, in the server configuration file[242], `server_config.xml`, change the **`https.enable`** setting to **`true`**. Modify the various HTTPS settings of the configuration file[242] according to your server requirements.

Private key and certificate

You can obtain a private key and certificate files in one of the following ways:

- From a certificate authority: Follow the steps described in the section Setting Up SSL Encryption[247].
- Create a self-signed certificate by using the following OpenSSL command (suitably modified for your environment):

```
openssl req -x509 -newkey rsa:4096 -nodes -keyout key.pem -out cert.pem -days 365 -
subj "/C=AT/ST=vienna/L=vienna/O=Altova Gmbh/OU=dev/CN=www.altova.com"
```

Testing the connection

A good way to test your connection is via the curl command line tool for transferring data with URLs. You can use the following command:

```
curl.exe https://localhost:443/v1/version
```

If the certificate is not trusted, use the **`-k`** option, like this:

```
curl.exe -k https://localhost:443/v1/version
```

The following command executes the HTTP Python example that is distributed with RaptorXML Server:

```
python3.exe examples\ServerAPI\python\RunRaptorXML.py --host localhost -p 443 -s
```

6.1.1.5 Setting Up SSL Encryption

If you wish to encrypt your RaptorXML Server data transfers using the SSL protocol, you will need to:

- Generate an SSL private key and create an SSL public key certificate file
- Set up RaptorXML Server for SSL communication.

The steps to do this are listed below.

This method uses the open-source OpenSSL toolkit to manage SSL encryption. The steps listed below, therefore, need to be carried out on a computer on which OpenSSL is available. OpenSSL typically comes pre-

installed on most Linux distributions and on macOS machines. It can also be installed on Windows computers. For download links to installer binaries, see the OpenSSL Wiki.

To generate a private key and obtain a certificate from a certificate authority, do the following:

1. Generate a private key

   SSL requires that a **private key** is installed on RaptorXML Server. This private key will be used to encrypt all RaptorXML Server data. To create the private key, use the following OpenSSL command:

   ```
   openssl genrsa -out private.key 2048
   ```

   This creates a file called **private.key**, which contains your private key. Note where you save the file. You will need the private key to (i) generate the Certificate Signing Request (CSR), and (ii) be installed on RaptorXML Server.

2. Certificate Signing Requests (CSRs)

   A Certificate Signing Request (CSR) is sent to a certificate authority (CA), such as VeriSign or Thawte, to request a public key certificate. The CSR is based on your private key and contains information about your organization. Create a CSR with the following OpenSSL command (which provides the private-key file, `private.key`, that was created in Step 1, as one of its parameters):

   ```
   openssl req -new -nodes -key private.key -out my.csr
   ```

   During generation of the CSR you will need to give information about your organization, such as that listed below. This information will be used by the certificate authority to verify your company's identity.

   - *Country*
   - *Locality* (the city where your business is located)
   - *Organization* (your company name). Do not use special characters; these will invalidate your certificate
   - *Common Name* (the DNS name of your server). This must exactly match your server's official name, that is, the DNS name client apps will use to connect to the server
   - *A challenge password.* Keep this entry blank!

3. Buy an SSL certificate

   Purchase an SSL certificate from a recognized certificate authority (CA), such as VeriSign or Thawte. For the rest of these instructions, we follow the VeriSign procedure. The procedure with other CAs is similar.

   - Go to the VeriSign website.
   - Click **Buy SSL Certificates**.
   - Different types of SSL certificates are available. For RaptorXML Server, Secure Site or Secure Site Pro certificates should be sufficient. EV (extended verification) is not necessary, since there is no "green address bar" for users to see.
   - Proceed through the sign-up process, and fill in the information required to place your order.
   - When prompted for the CSR (*created in Step 2*), copy and paste the content of the **my.csr** file into the order form.

 © 2017-2023 Altova GmbH

- Pay for the certificate with your credit card.

> **Allow time for obtaining a certificate**
> Obtaining public key certificates from an SSL certificate authority (CA) typically takes **two to three business days**. Please take this into account when setting up your RaptorXML Server.

4. Receive public key from CA

   Your certificate authority will complete the enrollment process over the next two to three business days. During this time you might get emails or phone calls to check whether you are authorized to request an SSL certificate for your DNS domain. Please work with the authority to complete the process.

   After the authorization and enrollment process has been completed, you will get an email containing the **public key** of your SSL certificate. The public key will be in plain text form or attached as a `.cer` file.

5. Save public key to file

   For use with RaptorXML Server, the public key must be saved in a `.cer` file. If the public key was supplied as text, copy-paste all the lines from

   ```
   --BEGIN CERTIFICATE--
     ...
   --END CERTIFICATE--
   ```

   into a text file that we will call `mycertificate.cer`.

6. Save CA's intermediate certificates to file

   To complete your SSL certificate, you will need two additional certificates: the **primary** and **secondary intermediate certificates**. Your certificate authority (CA) will list content of intermediate certificates on its website.

   - Verisign's intermediate certificates: https://knowledge.verisign.com/support/ssl-certificates-support/index?page=content&id=AR657&actp=LIST&viewlocale=en_US
   - Verisign's intermediate certificates for its Secure Site product: https://knowledge.verisign.com/support/ssl-certificates-support/index?page=content&id=AR1735

   Copy-paste both intermediate certificates (primary and secondary) into separate text files and save them on your computer.

7. Optionally combine certificates in one public key certificate file

   You now have three certificate files:

- Public key (`mycertificate.cer`)
- Secondary intermediate certificate
- Primary intermediate certificate

You can integrate your intermediate certificates into your public key certificate if you like. How to do this is described below. (Alternatively, you can use the **`https.certificate-chain`** configuration file setting[243] to specify the location of intermediate certificates.)

Each contains text blocks bracketed by lines that look like this:

```
--BEGIN CERTIFICATE--
  ...
--END CERTIFICATE--
```

Now copy-paste all three certificates into one file so that they are in sequence. The order of the sequence is important: (i) public key, (ii) secondary intermediate certificate, (iii) primary intermediate certificate. Ensure that there are no lines between certificates.

```
--BEGIN CERTIFICATE--
  public key from mycertificate.cer (see Step 5)
--END CERTIFICATE--
--BEGIN CERTIFICATE--
  secondary intermediate certificate (see Step 6)
--END CERTIFICATE--
--BEGIN CERTIFICATE--
  primary intermediate certificate (see Step 6)
--END CERTIFICATE--
```

Save the resulting combined certificate text to a file named **`publickey.cer`** . This is the *public key certificate file* of your SSL certificate. It includes your public key certificate as well as the complete chain of trust in the form of the intermediate certificates that were used by the CA to sign your certificate.

6.1.2 Client Requests

After RaptorXML Server has been started as a service[240], its functionality can be accessed by any HTTP client which can:

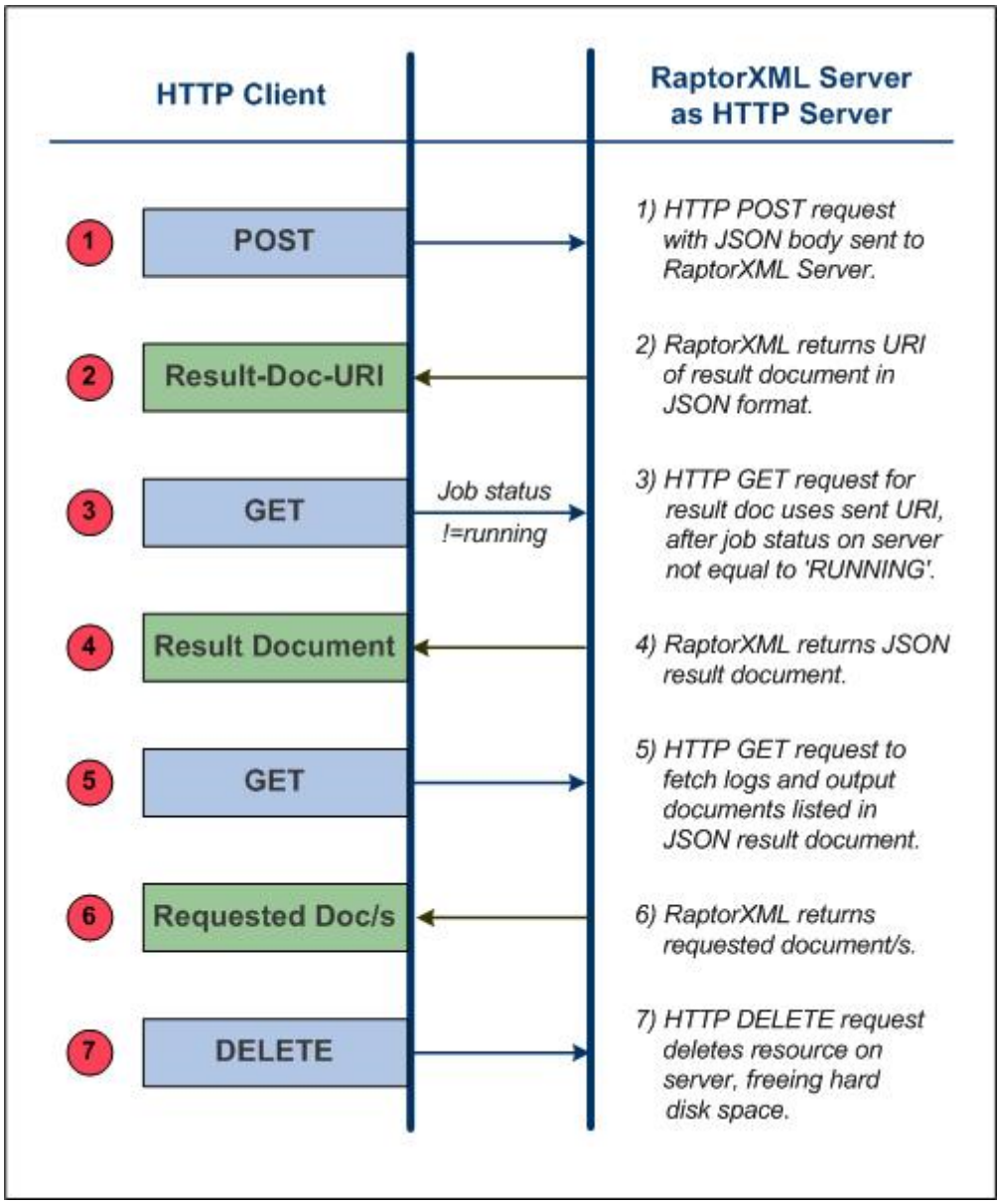
- use the HTTP methods `GET`, `PUT`, `POST`, and `DELETE`
- set the `Content-Type` header field

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.

© 2017-2023 Altova GmbH

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[252], with the body of the request being in **JSON format**. The request could be for any functionality of RaptorXML 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[56]. The request is posted to: `http://localhost:8087/v1/queue`, assuming `localhost:8087` is the address of RaptorXML Server (the initial address of the server[246]). Such a request is termed a **RaptorXML Server job**.

2. If the request is received and accepted for processing by RaptorXML 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[264]. 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[264] (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 Server returns the result document in JSON format[265]. The result document might contain the **URIs of error or output documents** produced by RaptorXML 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[268] received in Step 4 via an HTTP `GET` method to the server. Each request is sent in a separate `GET` method.

6. RaptorXML Server returns the requested documents[268] in response to the `GET` requests made in Step 5.

7. The client can delete unwanted documents on the server[269] 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.

6.1.2.1 Initiating Jobs with POST

This section:

- Sending the request[252]
- JSON syntax for POST requests[252]
- Uploading files with the POST request[254]
- Uploading ZIP archives[255]

Sending the request

A RaptorXML Server job is initiated with the HTTP `POST` method

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

 © 2017-2023 Altova GmbH

Note the following points:

- The URI above has a server address that uses the settings of the initial configuration[242].
- 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[254] 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[254]. 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[254] 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[56].

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[56] 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=---PartBoundary
```

 © 2017-2023 Altova GmbH

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.

See Example-1 (with Callouts): Validate XML 256 for a detailed explanation of the code, and Example-2: Using a Catalog to Find the Schema 257.

Testing with CURL

You can use a third-party data-transfer application such as CURL (http://curl.haxx.se/) to test the POST request. CURL provides a helpful trace option that generates and lists the part boundaries of the requests. This will save you the task of manually creating the part boundaries. How you can use CURL is described in the section, Testing with CURL 259.

Uploading ZIP archives

ZIP archives can also be uploaded, and files within a ZIP can be referenced by using the **additional-files** scheme. For example:

```
additional-files:///mybigarchive.zip%7Czip/biginstance.xml
```

Note: The `|zip/` part needs to be URI-escaped as `%7Czip/` in order to conform to the URI RFC since the pipe `|` symbol is not directly allowed. The use of glob patterns (`*` and `?`) is also allowed. So you can use something like this to validate all XML files within the ZIP archive:

```
{"command": "xsi", "args": ["additional-files:///mybigarchive.zip%7Czip/*.xml"],
"options": {...}}
```

See Example-3: Using ZIP Archives 258 for a listing of example code.

6.1.2.1.1 Example-1 (with Callouts): Validate XML

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 the following CLI equivalent:

```
raptorxml 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*[254] *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[254]. The part boundary separator must be a unique string that will not occur anywhere in the

embedded documents. It is prefixed with two dashes and is used to separate the multiple parts. 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[253]. 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. Note that the last part boundary separator is both prefixed and postfixed with two dashes.

6.1.2.1.2 Example-2: Using a Catalog to Find the Schema

In this example, a catalog file is used to find the XML schema that is referenced by the XML files to be validated.

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

{"command": "xsi", "args": ["additional-files:///First.xml", "additional-
files:///Second.xml"], "options": {"user-catalog": "additional-files:///catalog.xml"}}

-----PartBoundary
Content-Disposition: attachment; filename="First.xml"; name="additional-files"
Content-Type: application/octet-stream

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

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

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

-----PartBoundary
```

```
Content-Disposition: attachment; filename="Demo.xsd"; name="additional-files"
Content-Type: application/octet-stream

<?xml version="1.0" encoding="UTF-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
Content-Disposition: attachment; filename="catalog.xml"; name="additional-files"
Content-Type: application/octet-stream

<?xml version='1.0' encoding='UTF-8'?>
<catalog xmlns='urn:oasis:names:tc:entity:xmlns:xml:catalog'
xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xsi:schemaLocation='urn:oasis:names:tc:entity:xmlns:xml:catalog Catalog.xsd'>
                  <uri name="http://example.com/Demo.xsd" uri="additional-
files:///Demo.xsd"/>
</catalog>

-----PartBoundary--
```

6.1.2.1.3 Example-3: Using ZIP Archives

ZIP archives can also be uploaded, and files within a ZIP can be referenced by using the **additional-files** scheme. For example:

```
additional-files:///mybigarchive.zip%7Czip/biginstance.xml
```

Note: The **|zip/** part needs to be URI-escaped as **%7Czip/** in order to conform to the URI RFC since the pipe **|** symbol is not directly allowed. The use of glob patterns (***** and **?**) is also allowed. So you can use something like this to validate all XML files within the ZIP archive:
`{"command": "xsi", "args": ["additional-files:///mybigarchive.zip%7Czip/*.xml"], "options": {…}}`

Note: `'Content-Disposition: form-data'` is also valid, in addition to `'Content-Disposition: attachment'`. Since several tools generate `form-data` as content-disposition, the value `form-data` is accepted as valid.

Example: Validating all XML files in a ZIP archive

In this example, it is assumed that all schema references are relative paths and that all schemas are contained within the zip.

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

{"command": "xsi", "args": ["additional-files:///Demo.zip%7Czip/*.xml"], "options": {}}
```

 © 2017-2023 Altova GmbH

```
-----PartBoundary
Content-Disposition: attachment; filename="Demo.zip"; name="additional-files"
Content-Type: application/octet-stream

Binary content of Demo.zip archive

-----PartBoundary--
```

Example: Validating XML files in a ZIP archive containing references to external schemas

In this example, the XML files in a ZIP archive are validated using references to an external schema, which is provided in a second ZIP archive.

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

{"command": "xsi", "args": ["additional-files:///Instances.zip%7Czip/*.xml"], "options":
{"user-catalog": "additional-files:///Schemas.zip%7Czip/catalog.xml"}}

-----PartBoundary
Content-Disposition: attachment; filename="Instances.zip"; name="additional-files"
Content-Type: application/octet-stream

Binary content of Instances.zip archive

-----PartBoundary
Content-Disposition: attachment; filename="Schemas.zip"; name="additional-files"
Content-Type: application/octet-stream

Binary content of Schemas.zip archive

-----PartBoundary--
```

6.1.2.1.4 Testing with CURL

The third-party application CURL (http://curl.haxx.se/) is a command line utility that you can use to test the `POST` request. CURL provides a very useful trace option that generates and lists the part boundaries of requests, which you can use directly in your requests or as a reference.

Given below is a sample test scenario in which an XML file is validated against an XML Schema. We assume the following:

- the commands below are executed from the folder in which the files to be submitted for validation are located; (this enables us to write simple relative paths to these files). If you have installed Altova's XMLSpy application, the files used in this example can be found in the application's `Examples` folder, which is located by default at: `C:\Users\<username>\Documents\Altova\XMLSpy2024\Examples`
- RaptorXML Server is running locally on port 8087

For more information about the CURL command line options, see the CURL Help.

Call CURL with the validation command on Windows

```
[input: powershell]
    \path\to\curl.exe -F 'msg={\"command\": \"xsi\", \"args\":[\"additional-
    files:///PurchaseOrder.zip%7Czip/ipo.xml\"], \"options\":{}};type=application/json' -F
    "additional-files=@PurchaseOrder.zip;type=application/octet-stream"
    http://localhost:8087/v1/queue
```

Note: In powershell, if quotes occur within quotes, different types of quotes (single/double) must be used.

```
[input: cmd]
    \path\to\curl.exe -F "msg={\"command\": \"xsi\", \"args\":[\"additional-
    files:///PurchaseOrder.zip%7Czip/ipo.xml\"], \"options\":{}};type=application/json" -F
    "additional-files=@PurchaseOrder.zip;type=application/octet-stream"
    http://localhost:8087/v1/queue

[output]
    {"jobid": "058F9E97-CB95-43EF-AC0A-496CD3AC43A3", "result": "/v1/results/058F9E97-CB95-
    43EF-AC0A-496CD3AC43A3"}
```

Use the URL of "result" to fetch the result

```
[input]
    \path\to\curl.exe http://localhost:8087/v1/results/058F9E97-CB95-43EF-AC0A-496CD3AC43A3

[output]
    {"jobid":"058F9E97-CB95-43EF-AC0A-496CD3AC43A3","state":"OK","error":{},"jobs":
    [{"file":"additional-files:///PurchaseOrder.zip%7Czip/ipo.xml","jobid":"D4B91CB0-CF03-
    4D29-B563-B6506E123A06","output":{},"state":"OK","error":{}}]}
```

CURL's trace option

CURL has a trace option (`--trace-ascii`), which traces the HTTP traffic sent to and from the server. The option is very useful since it lists the part boundaries that are required for initiating jobs with POST. You can use the information in the trace, either directly or as a reference, to create the part boundaries. The listing below shows the trace obtained by running the command given above.

- *Trace listing*

```
== Info:   Trying ::1...
== Info: Connected to localhost (::1) port 8087 (#0)
=> Send header, 217 bytes (0xd9)
0000: POST /v1/queue HTTP/1.1
0019: Host: localhost:8087
002f: User-Agent: curl/7.42.1
0048: Accept: */*
0055: Content-Length: 2939
```

 © 2017-2023 Altova GmbH

```
006b: Expect: 100-continue
0081: Content-Type: multipart/form-data; boundary=--------------------
00c1: ----d887ed58324015c3
00d7:
<= Recv header, 23 bytes (0x17)
0000: HTTP/1.1 100 Continue
=> Send data, 393 bytes (0x189)
0000: --------------------------d887ed58324015c3
002c: Content-Disposition: form-data; name="msg"
0058: Content-Type: application/json
0078:
007a: {"command": "xsi", "args":["additional-files:///PurchaseOrder.zi
00ba: p%7Czip/ipo.xml"], "options":{}}
00dc: --------------------------d887ed58324015c3
0108: Content-Disposition: form-data; name="additional-files"; filenam
0148: e="PurchaseOrder.zip"
015f: Content-Type: application/octet-stream
0187:
=> Send data, 2498 bytes (0x9c2)
0000: PK........"..6}.c.....M.......ipo.xsd.T.N.@.}N....O 5v.}..S....(
0040:  .JU/...$Y..5{.E.•.....I*...g...Y...\....Z..~......P.A.ct....•y.
...
0940: ..........."..6]g......l.............  .......address.xsdPK.......
0980: ..."..6I..v..................  .......ipo.xmlPK..................
09c0: ..
=> Send data, 48 bytes (0x30)
0000:
0002: --------------------------d887ed58324015c3--
<= Recv header, 22 bytes (0x16)
0000: HTTP/1.1 201 Created
<= Recv header, 13 bytes (0xd)
0000: Allow: POST
<= Recv header, 32 bytes (0x20)
0000: Content-Type: application/json
<= Recv header, 37 bytes (0x25)
0000: Date: Fri, 24 Jul 2015 16:58:08 GMT
<= Recv header, 24 bytes (0x18)
0000: Server: CherryPy/3.6.0
<= Recv header, 21 bytes (0x15)
0000: Content-Length: 111
<= Recv header, 2 bytes (0x2)
0000:
<= Recv data, 111 bytes (0x6f)
0000: {"jobid": "058F9E97-CB95-43EF-AC0A-496CD3AC43A3", "result": "/v1
0040: /results/058F9E97-CB95-43EF-AC0A-496CD3AC43A3"}
== Info: Connection #0 to host localhost left intact
```

Note: Notice from the above listing that `'Content-Disposition: form-data'` is also valid, in addition to `'Content-Disposition: attachment'`.

Call CURL with the well-formed-check command on Linux

```
/path/to/curl -F 'msg={"command": "wfxml", "args":[]};type=application/json' -F
"args=@ipo.xml;type=application/octet-stream" http://localhost:8087/v1/queue

/path/to/curl -F 'msg={"command": "wfxml", "args":["additional-files:///ipo.zip%
7Czip/ipo.xml"]};type=application/json' -F "additional-
files=@ipo.zip;type=application/octet-stream" http://localhost:8087/v1/queue
```

6.1.2.2 Server Response to POST Request

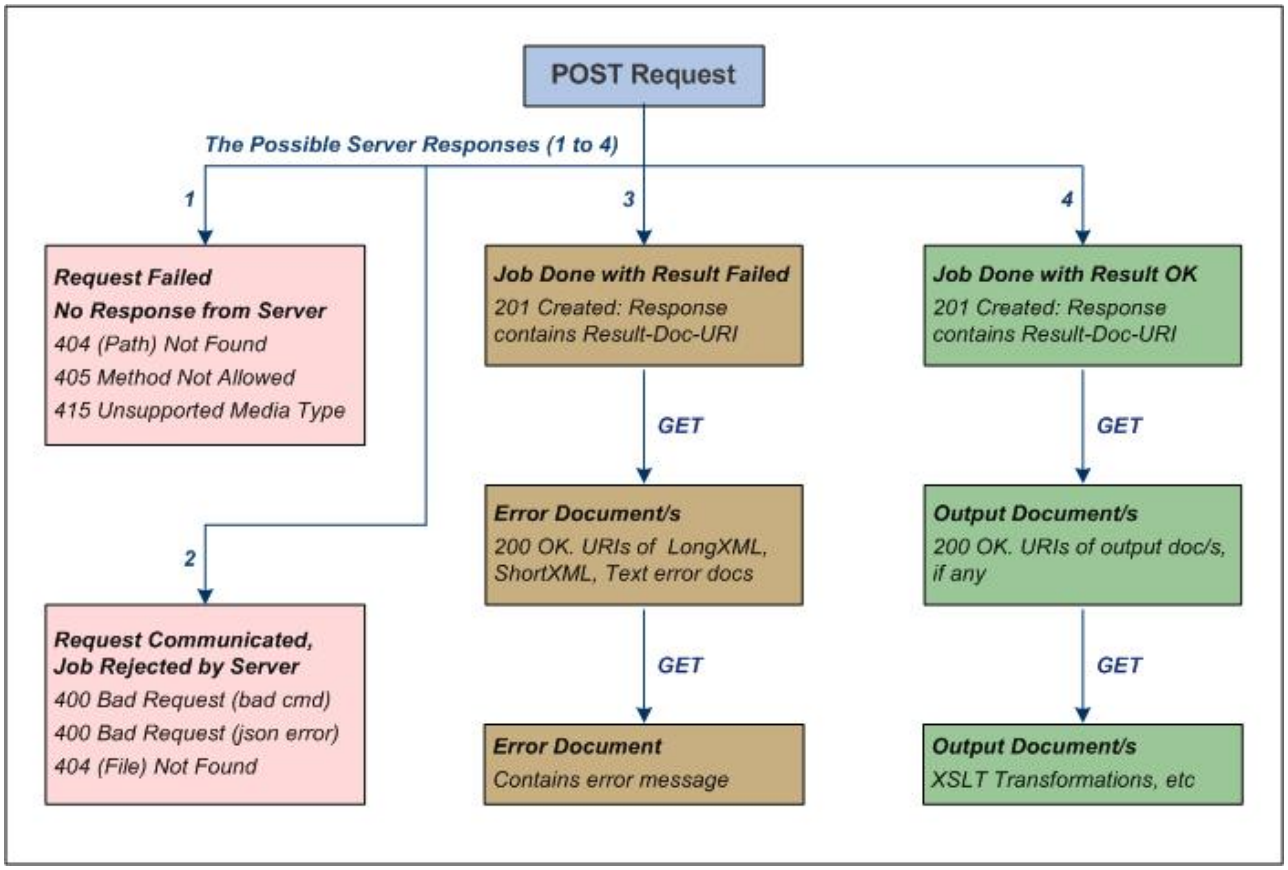
This section:

- Overview of possible server responses 262
- Response: Request failed, no response from server 263
- Response: Request communicated, but job rejected by server 263
- Response: Job executed (with positive or negative result) 264

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 meantime, if the result document is requested 264, a `"status": "Running"` message is returned if the job has been started but 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 been 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.

 © 2017-2023 Altova GmbH



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 56 is incorrect.
400 Bad Request (*json error*)	The request body has a JSON syntax 253 error.

404 File Not Found	Check file URI (or filepath) syntax 253 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 has been 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. After the result document has been created, it can be fetched with an HTTP `GET` request.

The result document itself might not yet have been created if processing of the job has not yet started or completed. If the result document is requested during this time, a `"status": "Running"` message is returned if the job has been started but has not been completed; a `Dispatched` state indicates that the job is in the server queue but has not yet been started.

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 268.
- *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 264 and Getting Error/Output Documents 268 for a description of these documents and how to access them.

6.1.2.3 Getting the Result Document

This section:

- The Result Document URI 264
- Fetching the Result Document 265
  - Result Document containing URIs of error documents 265
  - Result Document containing URIs of output documents 266
  - Result Document containing no URI 267
- Accessing error and output documents listed in the Result Document 268

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 [246]. For example, if the server address is `http://localhost:8087/` (the initial configuration address [242]), 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* [264]), 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 [238]. 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 [238]). For the details of these URIs, see the section Getting Error/Message/Output Documents [268].

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* 264).

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/1"
          ]
        },
        "state": "OK",
        "output-mapping":
        {
         "/v1/results/D34B5684-C6FF-4A7A-BF35-EBB9A8A8C2C8/output/1":
         "file:///c:/temp/test.html"
        },
        "error":
        {
```

 © 2017-2023 Altova GmbH

```
        }
      }
    ]
}
```

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* 264).
- 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.
- Alternatively, it is possible to retrieve all the generated result documents for a specific job as a zip archive using the URL `"/v1/results/`*JOBID*`/output/zip"`. This feature is not available in unrestricted filesystem mode. Please note that the zip archive will contain mangled file names, which need to be mapped back to the actual names using the `output-mapping` object.

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 268.

6.1.2.4 Getting Error/Message/Output Documents

A result document 264 can contain the file URIs or relative URIs of error documents 265, message documents (such as logs), and/or output documents 266. (There are some situations 267 in which a result document might not contain any URI.) The various kinds of URIs are described below 268.

To access these documents via HTTP, do the following:

1. Expand the relative URI 269 of the file in the result document to its absolute URI
2. Use the expanded URI in an HTTP GET request 269 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 243 (the default setting).
- To the job directory on the server. The server is configured 242 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` 243 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.* 266)

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:

 © 2017-2023 Altova GmbH

`/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*[266]). 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[264] 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[242])

and the relative URI of an error file in the result document[264] 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[242] and Getting the Result Document[264].

Use an HTTP `GET` request to access the file

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

6.1.2.5 Freeing Server Resources after Processing

RaptorXML 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[264] 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 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*[243] `] + [jobid`[264]`]`

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.

6.1.3 C# Example for REST API

Your RaptorXML Server installation contains a C# project that accesses RaptorXML Server's REST client interface to execute a set of jobs. The example project consists of two parts:

- **`RaptorXMLREST.cs`**: A wrapper class in C# that implements the REST mechanism to communicate with RaptorXML Server via HTTP.
- **`Program.cs`**: The C# program code that defines the jobs to be sent to RaptorXML Server via the REST wrapper.

These two parts are described in the subsections of this section: C# Wrapper for REST API 270 and Program Code for REST Requests 271.

Note that you can use any suitable REST wrapper for C# code. The main reason that we have created our own wrapper is so that the C# program code can be more tightly integrated with the wrapper class, thereby making an understanding of RaptorXML Server's REST interface easier.

Location and use of the C# example

The example project is located in the folder `C:\Program Files (x86)\Altova\RaptorXML Server2024\examples\REST_API\C#_RaptorREST_API`.

The example project was created using Visual Studio 2019, so you should use this version or later to build and run the project. Note that the C# example files are located in the Program Files folder, so you will need to open Visual Studio with administrator rights in order to access the files. Alternatively, you can copy the example project to another location and make relevant amendments to the project.

6.1.3.1 C# Wrapper for REST API

The wrapper class is defined in the C# file named **`RaptorXMLREST.cs`**, and it is named **`RaptorXMLRESTAPI`**.

It defines the following key classes for sending HTTP requests and receiving HTTP responses via REST:

- `Command`
- `MultiPartCommand`
- `CommandResponse`
- `ResultDocument`

It defines the following functions:

- `pollCommandResult`
- `fetchCommandResult`
- `sendRequest`
- `cleanupResults`

 © 2017-2023 Altova GmbH

To see how the wrapper implements the REST API, read the Client Requests[250] section to understand how the REST API works. After that you can read the C# code of the wrapper class to see how the wrapper implements C# code for the REST API.

For example, if you want to see how a command is sent to RaptorXML Server from C# code, you could do the following:

- The REST interface enables a command to be sent to RaptorXML Server via a HTTP **POST** request. This mechanism is described in the topic Initiating Jobs with POST[252].
- The next question is: How would the wrapper pass the command to the REST API? The mechanism for this is defined in the wrapper's **Command** class. Open the file **RaptorXMLREST.cs** to see the code of the **Command** class.
- Finally, to see how the program code[271] instantiates the wrapper's **Command** class, see the code of the three jobs in the program code[271].

6.1.3.2 Program Code for REST Requests

The C# program code containing the jobs for RaptorXML Server is defined in the C# file named **Program.cs**. The code uses the classes defined in the C# Wrapper for REST API[270] to create the REST requests that are sent to RaptorXML Server.

In the program code, there are three use cases to demonstrate how to use RaptorXML Server's REST API :

- Validation of a referenced XML file[271] with RaptorXML Server's **valany**[195] command. The schema file is referenced from within the XML file and does not need to be provided as an argument of the command.
- Two XML files are validated[272] using RaptorXML Server's **valany**[195] command. Both XML files, as well as the schema file used for the validation, are uploaded with the command as string attachments. The result of the validations are returned together after both validations have completed.
- An XML file is uploaded and transformed by an XSLT file[272]. Both files are uploaded via REST. The command used is RaptorXML Server's **xslt**[121]. The document resulting from the transformation is retrieved by the program..

The code for these three use cases is discussed in more detail below.

Error handling

In the event that an error is returned, an error handler function (named **HandleError**) at the bottom of the code retrieves the error message from the server response[262].

Case 1: Validate a referenced XML file (simple command)

The program code for this case uses classes and functions from the REST API wrapper to set up and execute the HTTP communication with RaptorXML Server. The logic of the code is as follows:

RaptorXMLRESTAPI.Command	Specifies the RaptorXML Server command to call, which is **valany**, and the file to be submitted as the argument of the `valany` command.

RaptorXMLRESTAPI.CommandResponse	Puts the server's response to the validation request into the **jsonResponse** variable. Note that validation jobs are reported as "OK" or "Failed" 264.
RaptorXMLRESTAPI.ResultDocument	Fetches the result document returned by the server and, if there are no errors, displays the validation result.

Case 2: Validate two uploaded XML files against an uploaded XSD (multipart command)

The program code for this case uses the **MultiPartCommand** class of the REST API wrapper to set up and execute the HTTP communication with RaptorXML Server. Since we want to upload files within the body of the POST request, the message header must have its content type set to multipart/form-data 252. The wrapper's **MultiPartCommand** class is used to set up the REST HTTP communication accordingly. The code for this use case is organized as follows:

RaptorXMLRESTAPI.MultiPartCommand	Specifies the RaptorXML Server command to call, which is **valany**, and then uses the **AppendAttachment** function of the class to upload the two XML files and the schema file. The files are submitted as strings. The server response returns the validation result of both files and this response is stored in the **jsonResponse** variable
RaptorXMLRESTAPI.fetchCommandResult	Fetches the result document returned by the server and, if there are no errors, displays the validation results.
RaptorXMLRESTAPI.cleanupResults	This function of the wrapper uses the **DELETE** method of HTTP to delete the result document file, temporary files, and error and output document files related to the job.

Case 3: XSLT transformation of uploaded XML and XSLT (multipart command)

The program code for this case is similar to that of Case 2 above. It uses the **MultiPartCommand** class to set up an XSLT transformation and display the result document in a message box. The XML and XSLT files for the transformation are uploaded with the request. Additionally, the **XSLT** command of RaptorXML Server also takes options, so this case shows how you could add options via the REST interface (in the example, this is done with the **RaptorXMLRESTAPI.AppendOption** function. Important points about the code are given below.

RaptorXMLRESTAPI.MultiPartCommand	Specifies the RaptorXML Server command to call, which is **XSLT**, and then uses (i) the **AppendAttachment** function of the class to upload the XML and XSLT files, and (ii) the **AppendOption** function to provide options for the RaptorXML Server command line. The uploaded files are submitted as strings. The server response returns the validation result of both files and this response is stored in the **jsonResponse** variable
RaptorXMLRESTAPI.fetchCommandResult	Fetches the result document returned by the server and, if there are no errors, displays the validation results.
RaptorXMLRESTAPI.cleanupResults	This function of the wrapper uses the **DELETE** method of HTTP to clean up the result document file, temporary files, and error and output

 © 2017-2023 Altova GmbH

	document files related to the job.

6.2 COM/.NET API

RaptorXML Server is licensed on the machine on which it is installed. The .NET interface is built as a wrapper around the COM interface. The COM and .NET interfaces of RaptorXML Server use a single API: the COM/.NET API of RaptorXML Server (object reference here 286).

You can use RaptorXML Server with:

- Scripting languages, such as JavaScript, via the COM interface
- Programming languages, such as C#, via the .NET Framework interface

6.2.1 COM Interface

RaptorXML Server is automatically registered as a COM server object when RaptorXML 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 Server installation package, it is best to de-install RaptorXML 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 **`RaptorXML.Server`** classes. These classes will typically be found under **`HKEY_LOCAL_MACHINE\SOFTWARE\Classes`**.

Code examples

- A VBScript example 274 showing how the RaptorXML API can be used via its COM interface is listed in the following topic.
- An example file corresponding to this listing is available in the `examples/API` folder of the RaptorXML application folder.

6.2.2 COM Example: VBScript

The VBScript example below is structured into the following parts:

- Set up and initialize the RaptorXML COM object 274
- Validate an XML file 275
- Perform an XSLT transformation, return the result as a string 275
- Process an XQuery document, save the result in a file 275
- Set up the execution sequence of the script and its entry point 276


```
' 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.InputFileName = "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
```

6.2.3 .NET Interface

The .NET interface is built as a wrapper around the RaptorXML Server COM interface. It is provided as a primary interop assembly signed by Altova; it uses the namespace `Altova.RaptorXMLServer`.

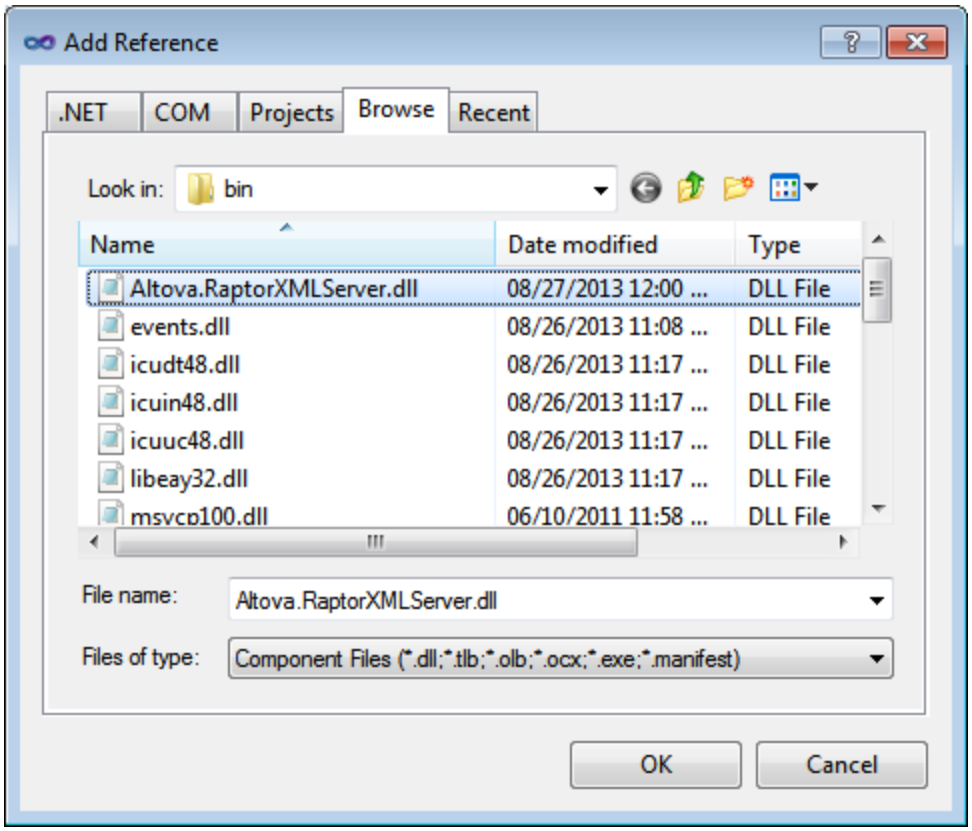
Adding the RaptorXML DLL as a reference to a Visual Studio .NET project

In order to use RaptorXML Server in your .NET project, add a reference to the RaptorXML DLL (`Altova.RaptorXMLServer.dll`) in your project. Your RaptorXML 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 Server is installed using the RaptorXML Server 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.

 © 2017-2023 Altova GmbH



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 277 and a Visual Basic .NET example 280 showing how the RaptorXML API can be used via its .NET interface are listed in the following topics. The files corresponding to these listings are available in the `examples/serverAPI` folder of the RaptorXML application folder.

6.2.4 .NET Example: C#

The C# example below does the following:

- Set up and initialize the RaptorXML .NET object 278
- Validate an XML file 278
- Perform an XSLT transformation, return the result as a string 278
- Process an XQuery document, save the result in a file 279
- Set up the execution sequence of the code and its entry point 279

```
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.InputFileName = "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 );
          }
```

 © 2017-2023 Altova GmbH

```
// 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() );
            }
        }
    }
}
```

6.2.5 .NET Example: Visual Basic .NET

The Visual Basic example below does the following:

- Set up and initialize the RaptorXML .NET object 280
- Validate an XML file 280
- Perform an XSLT transformation, return the result as a string 281
- Process an XQuery document, save the result in a file 281
- Set up the execution sequence of the code and its entry point 281

```
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.InputFileName = "MyXMLFile.xml"

       ' Validate; in case of invalid file report the problem returned by RaptorXML
```

 © 2017-2023 Altova GmbH

```
        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
```

© 2017-2023 Altova GmbH

6.3 Java API

The RaptorXML Server API can be accessed from Java code. To access RaptorXML Server from Java code, the libraries listed below must be listed 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.

6.3.1 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`[286] objects. So, a `RaptorXMLFactory` instance can be created with the call: `RaptorXML.getFactory()`.

The `RaptorXMLFactory`[286] interface provides methods for getting engine objects for validation and other processing functionality (such as XSLT transformation).

RaptorXMLFactory

The public `RaptorXMLFactory`[286] interface is described by the following listing:

```
public interface RaptorXMLFactory
{
   public XMLValidator[303] getXMLValidator();
   public XMLDSig[296] getXMLDSig();
   public XQuery[315] getXQuery();
   public XSLT[327] getXSLT();
   public void setServerName(String name) throws RaptorXMLException[295];
   public void setServerPath(String path) throws RaptorXMLException[295];
   public void setServerPort(int port) throws RaptorXMLException[295];
   public void setGlobalCatalog(String catalog);
   public void setUserCatalog(String catalog);
   public void setGlobalResourcesFile(String file);
   public void setGlobalResourceConfig(String config);
   public void setErrorFormat(RaptorXMLException[295] format);
   public void setErrorLimit(int limit);
   public void setReportOptionalWarnings(boolean report);
}
```

For more details, see the descriptions of `RaptorXMLFactory`[286] and the respective Java methods. Also see the Example Java Project[284].

6.3.2 Example Java Project

The Java 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 284
- Validate an XML file 284
- Perform an XSLT transformation, return the result as a string 284
- Process an XQuery document, return the result as a string 285
- Run the project 285

This basic functionality is included in the files in the `examples/API` folder of the RaptorXML 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.setInputFromText( "<!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" );
```

 © 2017-2023 Altova GmbH

```
        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();
        }

    }

}
```

6.4 Server API Reference

This section describes the RaptorXML Server API: its object model and the details of its interfaces and enumerations. The API description applies to both the COM/.NET and Java interfaces. While the structure of the API is the same for both interfaces, the names of methods and properties are different. For this reason, each method, property, and enumeration is described with a separate signature for COM/.NET and Java.

The starting point for using the functionality of RaptorXML Server is the IServer[286] interface (COM/.NET) or RaptorXMLFactory[286] class (Java).

6.4.1 Interfaces/Classes

The starting point for using the functionality of RaptorXML is the IServer[286] interface (COM/.NET) or RaptorXMLFactory[286] class (Java). This object contains the objects that provide the RaptorXML functionality: XML validation, XQuery document and XML Signature processing, and XSLT transformations.

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 (COM/.NET) / RaptorXMLFactory (Java)
  |-- IXMLDSig(COM/.NET) / XMLDSig (Java)
  |-- IXMLValidator (COM/.NET) / XMLValidator (Java)
  |-- IXSLT (COM/.NET) / XSLT (Java)
  |-- IXQuery (COM/.NET) / XQuery (Java)
```

6.4.1.1 IServer/RaptorXMLFactory

Use the **IServer/RaptorXMLFactory** interface to access the RaptorXML engine that you want. Note that the name of the interface in the COM/.NET API is different than that of the interface in the Java API:

- In COM/.NET: **IServer**
- In Java: **RaptorXMLFactory**

The methods and properties of **IServer/RaptorXMLFactory** are described in this section.

Java API entry-point method

The Java API is packaged in the **com.altova.raptorxml** package. The **RaptorXML** class provides an entry-point method called **getFactory()**, which returns RaptorXMLFactory[286] objects. So, a `RaptorXMLFactory` instance can be created with the call: `RaptorXML.getFactory()`.

© 2017-2023 Altova GmbH

6.4.1.1.1 Methods

The methods of the **IServer** (COM/.NET) and **RaptorXMLFactory** (Java) interfaces return an instance of the respective RaptorXML engine or class: XMLDSig, XML Validator, XSLT, and XQuery.

COM/.NET	Java
GetXMLDsig[287] *(for XML Signatures)*	getXMLDsig[287] *(for XML Signatures)*
GetXMLValidator[287]	getXMLValidator[287]
GetXQuery[288]	getXQuery[288]
GetXSLT[288]	getXSLT[288]

6.4.1.1.1.1 GetXMLDsig (for XML Signatures)

Returns an instance of the XML Signature interface/class (**XMLDSig**[296]).

COM and .NET

Signature: **IXMLDSig**[296] **GetXMLDSig()**

Java

Signature: **public XMLDSig**[296] **getXMLDSig()**

6.4.1.1.1.2 GetXMLValidator

Returns an instance of the XML Validator Engine.

COM and .NET

Signature: **IXMLValidator**[303] **GetXMLValidator()**

Java

Signature: **public XMLValidator**[303] **getXMLValidator()**

6.4.1.1.1.3 GetXQuery

Returns an instance of the XQuery Engine.

COM and .NET

Signature: **IXQuery**[315] **GetXQuery()**

Java

Signature: **public XQuery**[315] **getXQuery()**

6.4.1.1.1.4 GetXSLT

Returns an instance of the XSLT Engine.

COM and .NET

Signature: **IXSLT**[327] **GetXSLT()**

Java

Signature: **public XSLT**[327] **getXSLT()**

6.4.1.1.2 Properties

The properties of the **IServer** (COM/.NET) and **RaptorXMLFactory** (Java) interfaces are described in this section.

COM/.NET	Java
APIMajorVersion[289]	getAPIMajorVersion[289]
APIMinorVersion[289]	getAPIMinorVersion[289]
APIServicePackVersion[290]	getAPIServicePackVersion[290]
ErrorFormat[290]	setErrorFormat[290]
ErrorLimit[290]	setErrorLimit[290]
GlobalCatalog[291]	setGlobalCatalog[291]

© 2017-2023 Altova GmbH

GlobalResourceConfig 291	setGlobalResourceConfig 291
GlobalResourcesFile 291	setGlobalResourcesFile 291
Is64Bit 292	ss64Bit 292
MajorVersion 292	getMajorVersion 292
MinorVersion 292	getMinorVersion 292
ProductName 293	getProductName 293
ProductNameAndVersion 293	getProductNameAndVersion 293
ReportOptionalWarnings 293	setReportOptionalWarnings 293
ServerName 294	setServerName 294
ServerPath 294	setServerPath 294
ServerPort 294	setServerPort 294
ServicePackVersion 295	getServicePackVersion 295
UserCatalog 295	setUserCatalog 295

6.4.1.1.2.1 APIMajorVersion

Returns the major version of the API as an integer. The API major version can be different from the product's major version 292 if the API is connected to another server.

COM and .NET

Signature: `int APIMajorVersion()`

Java

Signature: `public int getAPIMajorVersion()`

6.4.1.1.2.2 APIMinorVersion

Returns the minor version of the API as an integer. The API minor version can be different from the product's minor version 292 if the API is connected to another server.

COM and .NET

Signature: `int APIMinorVersion()`

Java

Signature: `public int getAPIMinorVersion()`

6.4.1.1.2.3 APIServicePackVersion

Returns the service pack version of the API as an integer. The service pack version of the API can be different from the product's service pack version[295] if the API is connected to another server.

COM and .NET

Signature: `int APIServicePackVersion()`

Java

Signature: `public int getAPIServicePackVersion()`

6.4.1.1.2.4 ErrorFormat

Sets the RaptorXML error format to one of the `ENUMErrorFormat`[339] literals (Text, ShortXML, LongXML).

COM and .NET

Signature: `ErrorFormat(ENUMErrorFormat[339] format)`

Java

Signature: `public void setErrorFormat(ENUMErrorFormat[339] format)`

6.4.1.1.2.5 ErrorLimit

Sets the RaptorXML validation error limit. The `limit` parameter is of type `int` (Java), `uint` (COM/.NET), 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`.

COM and .NET

Signature: `ErrorLimit(uint limit)`

 © 2017-2023 Altova GmbH

Java

Signature: `public int setErrorLimit(int limit)`

6.4.1.1.2.6 GlobalCatalog

Sets the location, as a URL, of the main (entry-point) catalog file. The supplied string must be an absolute URL that gives the exact location of the main catalog file to use.

COM and .NET

Signature: `GlobalCatalog(string catalog)`

Java

Signature: `public void setGlobalCatalog(string catalog)`

6.4.1.1.2.7 GlobalResourceConfig

Sets the active configuration of the global resource. The `config` parameter is of type `String`, and specifies the name of the configuration used by the active global resource.

COM and .NET

Signature: `GlobalResourceConfig(string config)`

Java

Signature: `public void setGlobalResourceConfig(string config)`

6.4.1.1.2.8 GlobalResourcesFile

Sets the location, as a URL, of the Global Resources XML File. The supplied string must be an absolute URL that gives the exact location of the Global Resources XML File.

COM and .NET

Signature: `GlobalResourcesFile(string url)`

Java

Signature: `public void setGlobalResourcesFile(string url)`

6.4.1.1.2.9 Is64Bit

Checks if the application is a 64-bit executable. Returns boolean `true` if the application is 64 bit, `false` if it is not. *Example:* For `Altova RaptorXML Server 2024r2sp1(x64)`, returns `true`. If an error occurs, a **RaptorXMLException** 295 is raised.

COM and .NET

Signature: `boolean Is64Bit()`

Java

Signature: `public boolean is64Bit()`

6.4.1.1.2.10 MajorVersion

Returns the major version of the product as an integer. *Example:* For `Altova RaptorXML Server 2018r2sp1(x64)`, returns `20` (the difference between the major version (`2018`) and the initial year `1998`). If an error occurs, a **RaptorXMLException** 295 is raised.

COM and .NET

Signature: `int MajorVersion()`

Java

Signature: `public int getMajorVersion()`

6.4.1.1.2.11 MinorVersion

Returns the minor version of the product as an integer. *Example:* For `Altova RaptorXML Server 2024r2sp1(x64)`, returns `2` (from the minor version number `r2`). If an error occurs, a **RaptorXMLException** 295 is raised.

COM and .NET

© 2017-2023 Altova GmbH

Signature: `int MinorVersion()`

Java

Signature: `public int getMinorVersion()`

6.4.1.1.2.12 ProductName

Returns the name of the product as a string. *Example:* For `Altova RaptorXML Server 2024r2sp1(x64)`, returns `Altova RaptorXML Server`. If an error occurs, a **RaptorXMLException** 295 is raised.

COM and .NET

Signature: `string ProductName()`

Java

Signature: `public string getProductName()`

6.4.1.1.2.13 ProductNameAndVersion

Returns the product name, major version, minor version, and service pack version of the product as a string. *Example:* For `Altova RaptorXML Server 2024r2sp1(x64)`, returns `Altova RaptorXML Server 2024r2sp1(x64)`. If an error occurs, a **RaptorXMLException** 295 is raised.

COM and .NET

Signature: `string ProductNameAndVersion()`

Java

Signature: `public string getProductNameAndVersion()`

6.4.1.1.2.14 ReportOptionalWarnings

Enables/disables the reporting of warnings. A value of `true` enables warnings; `false` disables them.

COM and .NET

Signature: `ReportOptionalWarnings(boolean report)`

Java

Signature: `public void setReportOptionalWarnings(boolean report)`

6.4.1.1.2.15 ServerName

Sets the name of the HTTP server through which the connection to RaptorXML Server is made. The input parameter is a string that gives the name of the HTTP server. If an error occurs, a **RaptorXMLException** 295 is raised.

COM and .NET

Signature: `ServerName(string name)`

Java

Signature: `public void setServerName(string name)`

6.4.1.1.2.16 ServerPath

Specifies, in the form of a URL, the path to the HTTP server.

COM and .NET

Signature: `ServerPath(string path)`

Java

Signature: `public void setServerPath(string path)`

6.4.1.1.2.17 ServerPort

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. The input parameter is an integer that specifies the access port on the HTTP server. If an error occurs, a **RaptorXMLException** 295 is raised.

COM and .NET

© 2017-2023 Altova GmbH

Signature: `ServerPort(int port)`

Java

Signature: `public void setServerPort(int port)`

6.4.1.1.2.18 ServicePackVersion

Returns the service pack version of the product as an integer. *Example:* For `RaptorXML Server 2024r2sp1(x64)`, returns `1` (from the service pack version number `sp1`). If an error occurs, a **RaptorXMLException** 295 is raised.

COM and .NET

Signature: `int ServicePackVersion()`

Java

Signature: `public int getServicePackVersion()`

6.4.1.1.2.19 UserCatalog

Sets the location, as a URL, of the custom user catalog file. The supplied string must be an absolute URL that gives the exact location of the custom catalog file to use.

COM and .NET

Signature: `UserCatalog(string userCatalog)`

Java

Signature: `public void setUserCatalog(string userCatalog)`

6.4.1.2 RaptorXMLException

Generates an exception that contains information about an error that occurs during processing. The `message` parameter provides information about the error.

COM and .NET

Signature: `RaptorXMLException(string message)`

Java

Signature: `public void RaptorXMLException(string message)`

6.4.1.3 XMLDSig (for XML Signatures)

Methods of the **IXMLDSig/XMLDSig** interface/class can be used to sign XML documents, verify signed documents, update (with a new signature) previously signed documents that have been modified, and remove signatures.

Note that the name of the interface in the COM/.NET API is different than that of the class in the Java API:

- In COM/.NET: **IXMLDSig**
- In Java: **XMLDSig**

6.4.1.3.1 Methods

The methods of the **IXMLDSig** interface (COM/.NET) and **XMLDSig** class (Java) are described in this section.

6.4.1.3.1.1 ExecuteRemove

Removes the XML signature of the signed XML file, and saves the resulting unsigned document to an output location defined by `outputPath`, which is a string that provides the URL of the file location. The result is `true` on success, `false` on failure.

COM and .NET

Signature: `boolean ExecuteRemove(string outputPath)`

Java

Signature: `public boolean executeRemove(string outputPath)`

6.4.1.3.1.2 ExecuteSign

Signs the XML document according to the specified signing options (given in the `signatureType` and `canonicalizationMethod` parameters; *see the xmlsignature-sign CLI command 183 for available values*). The output file is defined by `outputPath`, which is a string that provides the URL of the output file. The result is `true` on success, `false` on failure.

 © 2017-2023 Altova GmbH

COM and .NET

Signature: `boolean ExecuteSign(string outputPath, string signatureType,string canonicalizationMethod)`

Java

Signature: `public boolean executeSign(string outputPath, string signatureType,string canonicalizationMethod)`

6.4.1.3.1.3 ExecuteUpdate

Updates the XML signature in the signed XML file. If the document has been modified, the updated XML signature will be different; otherwise, the updated signature will be the same as the previous signature. The output file is specified with `outputPath`, which is a string that provides the URL of the file with the updated signature. The result is `true` on success, `false` on failure.

Either (i) the HMAC secret key 301 property or (ii) the certificate-name 298 and certificate-store 299 properties must be specified. If the certificate options are specified, then they must match those that were used to sign the XML document previously. (Note that the certificate-store option is currently not supported on Linux and macOS.)

COM and .NET

Signature: `boolean ExecuteUpdate(string outputPath)`

Java

Signature: `public boolean executeUpdate(string outputPath)`

6.4.1.3.1.4 ExecuteVerify

Returns the result of the signature verification: **`true`** if verification is successful, **`false`** otherwise.

COM and .NET

Signature: `boolean ExecuteVerify()`

Java

Signature: `public boolean executeVerify()`

6.4.1.3.2 Properties

The properties of the **IXMLDSig** interface (COM/.NET) and **XMLDSig** class (Java) are described in this section.

6.4.1.3.2.1 AbsoluteReferenceUri

Specifies whether the URI of the signed document is to be read as absolute (`true`) or relative (`false`). Default is `false`.

COM and .NET

Signature: **AbsoluteReferenceUri(boolean absoluteuri)**

Java

Signature: **public** void **setAbsoluteReferenceUri(boolean absoluteuri)**

6.4.1.3.2.2 AppendKeyInfo

Specifies whether to include the `KeyInfo` element in the signature or not. The default is `false`.

COM and .NET

Signature: **AppendKeyInfo(boolean include)**

Java

Signature: **public** void **setAppendKeyInfo(boolean inlude)**

6.4.1.3.2.3 CertificateName

The name of the certificate used for signing.

Windows
This is the **Subject** name of a certificate from the selected **--certificate-store**.

Example to list the certificates (under PowerShell)

```
% ls cert://CurrentUser/My
```

 © 2017-2023 Altova GmbH

```
PSParentPath: Microsoft.PowerShell.Security\Certificate::CurrentUser\My
Thumbprint Subject
---------- -------
C9DF64BB0AAF5FA73474D78B7CCFFC37C95BFC6C CN=certificate1
... CN=...
```

Example: **--certificate-name==certificate1**

Linux/MacOS
--certname specifies the file name of a PEM encoded X.509v3 certificate with the private key. Such files usually have the extension **.pem**.

Example: **--certificate-name==/path/to/certificate1.pem**

COM and .NET

Signature: **CertificateName(string name)**

Java

Signature: **public** void **setCertificateName(string name)**

6.4.1.3.2.4 CertificateStore

The location where the certificate specified with **--certificate-name** is stored.

Windows
The name of a certificate store under **cert://CurrentUser**. The available certificate stores can be listed (under PowerShell) by using **% ls cert://CurrentUser/**. Certificates would then be listed as follows:

```
Name : TrustedPublisher
Name : ClientAuthIssuer
Name : Root
Name : UserDS
Name : CA
Name : ACRS
Name : REQUEST
Name : AuthRoot
Name : MSIEHistoryJournal
Name : TrustedPeople
Name : MyCertStore
Name : Local NonRemovable Certificates
Name : SmartCardRoot
Name : Trust
Name : Disallowed
```

Example: **--certificate-store==MyCertStore**

Linux/MacOS
The `--certstore` option is currently not supported.

COM and .NET

Signature: `CertificateStore(string filelocation)`

Java

Signature: `public void setCertificateStore(string filelocation)`

6.4.1.3.2.5 DigestMethod

The algorithm that is used to compute the digest value over the input XML file. Available values are: `sha1|sha256|sha384|sha512`.

COM and .NET

Signature: `DigestMethod(string algo)`

Java

Signature: `public void setDigestMethod(string algo)`

6.4.1.3.2.6 HMACOutputLength

Truncates the output of the HMAC algorithm to `length` bits. If specified, this value must be

- a multiple of 8
- larger than 80
- larger than half of the underlying hash algorithm's output length

COM and .NET

Signature: `HMACOutputLength(int length)`

Java

Signature: `public void setHMACOutputLength(int length)`

© 2017-2023 Altova GmbH

6.4.1.3.2.7 HMACSecretKey

The HMAC shared secret key; must have a minimum length of six characters.

COM and .NET

Signature: `HMACSecretKey(string key)`

Java

Signature: `public void setHMACSecretKey(string key)`

6.4.1.3.2.8 InputXMLFileName

Sets the location, as a URL, of the XML document to process. The supplied string must be an absolute URL that gives the exact location of the XML file.

COM and .NET

Signature: `InputXMLFileName(string filepath)`

Java

Signature: `public void setInputXMLFileName(string filepath)`

6.4.1.3.2.9 LastErrorMessage

Retrieves a string that is the last error message from the RaptorXML engine.

COM and .NET

Signature: `string LastErrorMessage()`

Java

Signature: `public string getLastErrorMessage()`

6.4.1.3.2.10 SignatureMethod

Specifies the algorithm to use for generating the signature.

When a certificate is used
If a certificate is specified, then `SignatureMethod` is optional and the value for this parameter is derived from the certificate. If specified, it must match the algorithm used by the certificate. Example: `rsa-sha256`.

When --hmac-secret-key is used
When `HMACSecretKey` is used, then `SignatureMethod` is mandatory. The value must be one of the supported HMAC algorithms:

- `hmac-sha256`
- `hmac-sha386`
- `hmac-sha512`
- `hmac-sha1` (discouraged by the specification)

Example: **`hmac-sha256`**

COM and .NET

Signature: **`SignatureMethod(string algo)`**

Java

Signature: **`public void setSignatureMethod(string algo)`**

6.4.1.3.2.11 Transforms

Specifies the XML Signature transformations applied to the input document. The supported values are:

- `REC-xml-c14n-20010315` for Canonical XML 1.0 (omit comments)
- `xml-c14n11` for Canonical XML 1.1 (omit comments)
- `xml-exc-c14n#` for Exclusive XML Canonicalization 1.0 (omit comments)
- `REC-xml-c14n-20010315#WithComments` for Canonical XML 1.0 (with comments)
- `xml-c14n11#WithComments` for Canonical XML 1.1 (with comments)
- `xml-exc-c14n#WithComments` for Exclusive XML Canonicalization 1.0 (with comments)
- `base64`
- `strip-whitespaces` Altova extension

COM and .NET

Signature: **`Transforms(string value)`**

© 2017-2023 Altova GmbH

Java

Signature: `public void setTransforms(string value)`

6.4.1.3.2.12 WriteDefaultAttributes

Specifies whether to include default attribute values from the DTD in the signed document.

COM and .NET

Signature: `WriteDefaultAttributes(boolean write)`

Java

Signature: `public void setWriteDefaultAttributes(boolean write)`

6.4.1.4 XMLValidator

The **`IXMLValidator/XMLValidator`** interface/class provides methods to (i) validate various types of documents, (ii) check documents for well-formedness, and (iii) extract an Avro schema from an Avro binary. You can also provide additional processing via a Python script.

Note that the name of the interface in the COM/.NET API is different than that of the class in the Java API:

- In COM/.NET: **`IXMLValidator`**
- In Java: **`XMLValidator`**

6.4.1.4.1 Methods

The methods of the **`IXMLValidator`** interface (COM/.NET) and **`XMLValidator`** class (Java) are described in this section.

6.4.1.4.1.1 AddPythonScriptFile

Specifies the Python script file that provides additional processing of the file submitted for validation. The supplied string must be an absolute URL of the Python script. The Python script will be processed with a Python package that is bundled with RaptorXML Server. The bundled Python package is version 3.11.5.

COM and .NET

Signature: **AddPythonScriptFile(string filepath)**

Java

Signature: **public void addPythonScriptFile(string filepath)**

6.4.1.4.1.2 ClearPythonScriptFile

Clears Python script files added with the **AddPythonScriptFile** method or **PythonScriptFile** property.

COM and .NET

Signature: **ClearPythonScriptFile()**

Java

Signature: **public void clearPythonScriptFile()**

6.4.1.4.1.3 ExtractAvroSchema

Extracts an Avro schema from a binary file. The `outputPath` parameter is an absolute URL that specifies the output location. The result is `true` on success, `false` on failure. If an error occurs, a **RaptorXMLException** 295 is raised. Use `LastErrorMessage` to access additional information.

COM and .NET

Signature: **ExtractAvroSchema(string outputPath)**

Java

Signature: **public void extractAvroSchema(string outputPath)**

6.4.1.4.1.4 IsValid

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 342 literal as its value. The result is `true` on success, `false` on failure. If an error occurs, a **RaptorXMLException** 295 is raised. Use `LastErrorMessage` to access additional information.

 © 2017-2023 Altova GmbH

COM and .NET

Signature: `boolean` **`IsValid(`**ENUMValidationType[342]**`type)`**

Java

Signature: **`public`** `boolean` **`isValid(`**ENUMValidationType[342]**`type)`**

6.4.1.4.1.5 IsWellFormed

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[344] literal as its value. The result is `true` on success, `false` on failure. If an error occurs, a **RaptorXMLException**[295] is raised. Use `LastErrorMessage` to access additional information.

COM and .NET

Signature: `boolean` **`isWellFormed(`**ENUMWellformedCheckType[342]**`type)`**

Java

Signature: **`public`** `boolean` **`isWellFormed(`**ENUMWellformedCheckType[342]**`type)`**

6.4.1.4.2 Properties

The properties of the **`IXMLValidator`** interface (COM/.NET) and **`XMLValidator`** class (Java) are described in this section.

6.4.1.4.2.1 AssessmentMode

Sets the assessment mode of the XML validation (`Strict/Lax`), which is given by an ENUMAssessmentMode[338] literal.

COM and .NET

Signature: **`AssessmentMode(`**ENUMAssessmentMode[338]**`mode)`**

Java

Signature: `public void setAssessmentMode(ENUMAssessmentMode mode)`

6.4.1.4.2.2 AvroSchemaFileName

Sets the location, as a URL, of the external Avro Schema to use. The supplied string must be an absolute URL that gives the exact location of the Avro Schema file.

COM and .NET

Signature: `AvroSchemaFileName(string url)`

Java

Signature: `public void setAvroSchemaFileName(string url)`

6.4.1.4.2.3 AvroSchemaFromText

Supplies a string that is the text content of the Avro Schema document to use.

COM and .NET

Signature: `AvroSchemaFromText(string avroschema)`

Java

Signature: `public void setAvroSchemaFromText(string avroschema)`

6.4.1.4.2.4 DTDFileName

Sets the location, as a URL, of the DTD document to use for validation. The supplied string must be an absolute URL that gives the exact location of the DTD document.

COM and .NET

Signature: `DTDFileName(string url)`

Java

Signature: `public void setDTDFileName(string url)`

© 2017-2023 Altova GmbH

6.4.1.4.2.5 DTDFromText

Supplies a string that is the text content of the DTD document to use for validation.

COM and .NET

Signature: `DTDFromText(string dtdtext)`

Java

Signature: `public void setDTDFromText(string dtdtext)`

6.4.1.4.2.6 EnableNamespaces

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. The default is `false`.

COM and .NET

Signature: `EnableNamespaces(boolean enableNS)`

Java

Signature: `public void setEnableNamespaces(boolean enableNS)`

6.4.1.4.2.7 InputFileArray

Provides an array of URLs of the files to be used as input data. The array is an object containing the strings of the absolute URLs of each of the input files.

COM and .NET

Signature: `InputFileArray(object fileArray)`

Java

Signature: `public void setInputFileArray(object fileArray)`

6.4.1.4.2.8 InputFileName

Sets the location, as a URL, of the input data file to process. The supplied string must be an absolute URL that gives the location of the input file.

COM and .NET

Signature: `InputFileName(string filepath)`

Java

Signature: `public void setInputFileName(string filepath)`

6.4.1.4.2.9 InputFromText

Supplies a string that is the text content of the document to process.

COM and .NET

Signature: `InputFromText(string doc)`

Java

Signature: `public void setInputFromText(string doc)`

6.4.1.4.2.10 InputTextArray

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.

COM and .NET

Signature: `InputTextArray(object textfileArray)`

Java

Signature: `public void setInputTextArray(object textfileArray)`

 © 2017-2023 Altova GmbH

6.4.1.4.2.11 InputXMLFileName

Sets the location, as a URL, of the XML document to process. The supplied string must be an absolute URL that gives the exact location of the XML file.

COM and .NET

Signature: `InputXMLFileName(string url)`

Java

Signature: `public void setInputXMLFileName(string url)`

6.4.1.4.2.12 InputXMLFromText

Supplies a string that is the text content of the XML document to process.

COM and .NET

Signature: `InputXMLFromText(string xml)`

Java

Signature: `public void setInputXMLFromText(string xml)`

6.4.1.4.2.13 Json5

If set to `true`, enables JSON 5 support.

COM and .NET

Signature: `Json5(boolean json5)`

Java

Signature: `public void setJson5(boolean json5)`

6.4.1.4.2.14 JSONSchemaFileName

Sets the location, as a URL, of the JSON Schema file that will be used for JSON instance-document validation. The supplied string must be an absolute URL that gives the exact location of the JSON Schema file.

COM and .NET

Signature: `JSONSchemaFileName(string url)`

Java

Signature: `public void setJSONSchemaFileName(string url)`

6.4.1.4.2.15 JSONSchemaFromText

Supplies a string that is the text content of the JSON Schema document that will be used for validation of the JSON instance document.

COM and .NET

Signature: `JSONSchemaFromText(string jsonschema)`

Java

Signature: `public void setJSONSchemaFromText(string jsonschema)`

6.4.1.4.2.16 LastErrorMessage

Retrieves a string that is the last error message from the RaptorXML engine.

COM and .NET

Signature: `string LastErrorMessage()`

Java

Signature: `public string getLastErrorMessage()`

 © 2017-2023 Altova GmbH

6.4.1.4.2.17 ParallelAssessment

Enables/disables parallel schema validity assessment[223].

COM and .NET

Signature: `ParallelAssessment(boolean enable)`

Java

Signature: `public void setParallelAssessment(boolean enable)`

6.4.1.4.2.18 PythonScriptFile

Specifies the Python script file that provides additional processing of the file submitted for validation. The supplied string must be an absolute URL of the Python script. The Python script will be processed with a Python package that is bundled with RaptorXML Server. The bundled Python package is version 3.11.5.

COM and .NET

Signature: `PythonScriptFile(string filepath)`

Java

Signature: `public void setPythonScriptFile(string filepath)`

6.4.1.4.2.19 SchemaFileArray

Supplies the collection of XML Schema files that will be used as external XML Schemas. The files are identified by their URLs. The input is a collection of strings, each of which is the absolute URL of an XML Schema file

COM and .NET

Signature: `SchemaFileArray(object urlArray)`

Java

Signature: `public void setSchemaFileArray(object urlArray)`

6.4.1.4.2.20 SchemaFileName

Sets the location, as a URL, of the XML Schema document to be used for validation. The supplied string must be an absolute URL that gives the exact location of the XML Schema file.

COM and .NET

Signature: `SchemaFileName(string filepath)`

Java

Signature: `public void setSchemaFileName(string filepath)`

6.4.1.4.2.21 SchemaFromText

Supplies a string that is the text content of the XML Schema document to use for validation of the XML instance document.

COM and .NET

Signature: `SchemaFileName(string xsdText)`

Java

Signature: `public void setSchemaFileName(string xsdText)`

6.4.1.4.2.22 SchemaImports

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 submitted.

COM and .NET

Signature: `SchemaImports(ENUMSchemaImports 340 importOption)`

Java

Signature: `public void setSchemaImports(ENUMSchemaImports 340 importOption)`

 © 2017-2023 Altova GmbH

6.4.1.4.2.23 SchemalocationHints

Specifies the mechanism to use to locate the schema. The mechanism is specified by the `ENUMLoadSchemalocation` literal that is selected.

COM and .NET

Signature: **SchemalocationHints(**ENUMLoadSchemalocation[339] **hint)**

Java

Signature: **public** void **setSchemalocationHints(**ENUMLoadSchemalocation[339] **hint)**

6.4.1.4.2.24 SchemaMapping

Sets what mapping to use in order to locate the schema. The mapping is specified by the `ENUMSchemaMapping` literal that is selected.

COM and .NET

Signature: **SchemaMapping(**ENUMSchemaMapping[342] **mappingOption)**

Java

Signature: **public** void **setSchemaMapping(**ENUMSchemaMapping[342] **mappingOption)**

6.4.1.4.2.25 SchemaTextArray

Supplies the content of multiple XML Schema files. The input is a collection of strings, each of which is the content of an XML Schema document.

COM and .NET

Signature: **SchemaTextArray(**object **schemaDocs)**

Java

Signature: **public** void **setSchemaTextArray(**object **schemaDocs)**

6.4.1.4.2.26 Streaming

Enables streaming validation. In streaming mode, data that is stored in memory is minimized and processing is faster. A value of `true` enables streaming; `false` disables it. Default is `true`.

COM and .NET

Signature: **`Streaming(boolean enable)`**

Java

Signature: **`public void setStreaming(boolean enable)`**

6.4.1.4.2.27 XincludeSupport

Enables or disables the use of `XInclude` elements. A value of `true` enables XInclude support; `false` disables it. The default value is `false`.

COM and .NET

Signature: **`XincludeSupport(boolean xinclude)`**

Java

Signature: **`public void setXincludeSupport(boolean xinclude)`**

6.4.1.4.2.28 XMLValidationMode

Sets the XML validation mode, which is an enumeration literal of ENUMXMLValidationMode[344] that determines whether to check validity or well-formedness.

COM and .NET

Signature: **`XMLValidationMode(`**ENUMXMLValidationMode[344] **`valMode)`**

Java

Signature: **`public void setXMLValidationMode(`**ENUMXMLValidationMode[344] **`valMode)`**

© 2017-2023 Altova GmbH

6.4.1.4.2.29 XSDVersion

Sets the XML Schema version against which the XML document will be validated. Value is an enumeration literal of ENUMXSDVersion[347].

COM and .NET

Signature: **XSDVersion(**ENUMXSDVersion[347] **version)**

Java

Signature: **public** void **setXSDVersion(**ENUMXSDVersion[347] **version)**

6.4.1.5 XQuery

The **IXQuery/XQuery** interface/class provides methods to (i) execute XQuery documents and XQuery updates, and (ii) validate XQuery-related documents. You can also provide data for the executions via external variables.

Note that the name of the interface in the COM/.NET API is different than that of the class in the Java API:

- In COM/.NET: **IXQuery**
- In Java: **XQuery**

6.4.1.5.1 Methods

The methods of the **IXQuery** interface (COM/.NET) and **XQuery** class (Java) are described in this section.

6.4.1.5.1.1 AddExternalVariable

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. The `name` parameter holds the name of the variable, which is a QName, as a string. The `value` parameter holds the value of the variable as a string.

COM and .NET

Signature: **AddExternalVariable(**string **name,** string **value)**

Java

Signature: `public void addExternalVariable(string name, string value)`

6.4.1.5.1.2 ClearExternalVariableList

Clears the external variables list created by the AddExternalVariable[315] method.

COM and .NET

Signature: `ClearExternalVariableList()`

Java

Signature: `public void clearExternalVariableList()`

6.4.1.5.1.3 Execute

Executes the XQuery transformation according to the XQuery version named in the EngineVersion[319] property, and saves the result to the output file named in the `outputFile` parameter. The parameter is a string that provides the location (path and filename) of the output file. The result is `true` on success, `false` on failure. If an error occurs, a RaptorXMLException[295] is raised. Use the `LastErrorMessage` property to access additional information.

COM and .NET

Signature: `boolean Execute(string outputFile)`

Java

Signature: `public boolean Execute(string outputFile)`

6.4.1.5.1.4 ExecuteAndGetResultAsString

Executes the XQuery update according to the XQuery Update specification named in the EngineVersion[319] property, and returns the result as a string. This method does not produce additional result files, such as charts or secondary results. It also does not hold binary results such as `.docx` OOXML files. If additional output files are needed, use the Execute[316] method.

 © 2017-2023 Altova GmbH

COM and .NET

Signature: `string` **`ExecuteAndGetResultAsString()`**

Java

Signature: **`public`** `string` **`executeAndGetResultAsString()`**

6.4.1.5.1.5 ExecuteUpdate

Executes the XQuery update according to the XQuery Update specification named in the `XQueryUpdateVersion`[326] property, and saves the result to the output file named in the `outputFile` parameter. The parameter is a string that provides the location (path and filename) of the output file. The result is `true` on success, `false` on failure. If an error occurs, a `RaptorXMLException`[295] is raised. Use the `LastErrorMessage` property to access additional information.

COM and .NET

Signature: `boolean` **`ExecuteUpdate(`**`string` **`outputFile)`**

Java

Signature: **`public`** `boolean` **`executeUpdate(`**`string` **`outputFile)`**

6.4.1.5.1.6 ExecuteUpdateAndGetResultAsString

Executes the XQuery update according to the XQuery Update specification named in the `XQueryUpdateVersion`[326] property, and returns the result as a string. This method does not produce additional result files, such as charts or secondary results. It also does not hold binary results such as `.docx` OOXML files.

COM and .NET

Signature: `string` **`ExecuteUpdateAndGetResultAsString()`**

Java

Signature: **`public`** `string` **`executeUpdateAndGetResultAsString()`**

6.4.1.5.1.7 IsValid

Returns the result of validating the XQuery document according to the XQuery specification named in the EngineVersion[319] property. The result is `true` on success, `false` on failure. If an error occurs, a RaptorXMLException[295] is raised. Use the `LastErrorMessage` property to access additional information.

COM and .NET

Signature: `boolean` **`IsValid()`**

Java

Signature: **`public`** `boolean` **`isValid()`**

6.4.1.5.1.8 IsValidUpdate

Returns the result of validating the XQuery Update document according to the XQuery Update specification named in the XQueryUpdateVersion[326] property. The result is `true` on success, `false` on failure. If an error occurs, a RaptorXMLException[295] is raised. Use the `LastErrorMessage` property to access additional information.

COM and .NET

Signature: `boolean` **`IsValidUpdate()`**

Java

Signature: **`public`** `boolean` **`isValidUpdate()`**

6.4.1.5.2 Properties

The properties of the **`IXQuery`** interface (COM/.NET) and **`XQuery`** class (Java) are described in this section.

6.4.1.5.2.1 AdditionalOutputs

Returns the additional outputs of the last executed job.

COM and .NET

© 2017-2023 Altova GmbH

Signature: `string` **`AdditionalOutputs()`**

Java

Signature: `public string` **`getAdditionalOutputs()`**

6.4.1.5.2.2 ChartExtensionsEnabled

Enables or disables Altova's chart extension functions. A value of `true` enables chart extensions; `false` disables them. Default value is `true`.

COM and .NET

Signature: **`ChartExtensionsEnabled(`**`boolean` **`enable)`**

Java

Signature: `public void` **`setChartExtensionsEnabled(`**`boolean` **`enable)`**

6.4.1.5.2.3 DotNetExtensionsEnabled

Enables or disables .NET extension functions. A value of `true` enables .NET extensions; `false` disables them. Default value is `true`.

COM and .NET

Signature: **`DotNetExtensionsEnabled(`**`boolean` **`enable)`**

Java

Signature: `public void` **`setDotNetExtensionsEnabled(`**`boolean` **`enable)`**

6.4.1.5.2.4 EngineVersion

Specifies the XQuery version to use. The property value is an ENUMXQueryVersion[346] literal.

COM and .NET

Signature: **`EngineVersion(`**ENUMXQueryVersion[346] **`version)`**

Java

Signature: `public void setEngineVersion(ENUMXQueryVersion 346 version)`

6.4.1.5.2.5 IndentCharacters

Submits the character string that will be used as indentation in the output.

COM and .NET

Signature: `IndentCharacters(string indentChars)`

Java

Signature: `public void setIndentCharacters(string indentChars)`

6.4.1.5.2.6 InputXMLFileName

Sets the location, as a URL, of the XML document to process. The supplied string must be an absolute URL that gives the exact location of the XML file.

COM and .NET

Signature: `InputXMLFileName(string url)`

Java

Signature: `public void setInputXMLFileName(string url)`

6.4.1.5.2.7 InputXMLFromText

Supplies a string that is the text content of the XML document to process.

COM and .NET

Signature: `InputXMLFromText(string xml)`

 © 2017-2023 Altova GmbH

Java

Signature: `public void setInputXMLFromText(string xml)`

6.4.1.5.2.8 JavaBarcodeExtensionLocation

Specifies the location of the barcode extension file. See the section on Altova's barcode extension functions [482] for more information. The supplied string must be an absolute URL that gives the base location of the file to use.

COM and .NET

Signature: `JavaBarcodeExtensionLocation(string url)`

Java

Signature: `public void setJavaBarcodeExtensionLocation(string url)`

6.4.1.5.2.9 JavaExtensionsEnabled

Enables or disables Java extension functions. A value of `true` enables Java extensions; `false` disables them. Default value is `true`.

COM and .NET

Signature: `JavaExtensionsEnabled(boolean enable)`

Java

Signature: `public void setJavaExtensionsEnabled(boolean enable)`

6.4.1.5.2.10 KeepFormatting

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`.

COM and .NET

Signature: `KeepFormatting(boolean keep)`

Java

Signature: `public void setKeepFormatting(boolean keep)`

6.4.1.5.2.11 LastErrorMessage

Retrieves a string that is the last error message from the RaptorXML engine.

COM and .NET

Signature: `string LastErrorMessage()`

Java

Signature: `public string getLastErrorMessage()`

6.4.1.5.2.12 LoadXMLWithPSVI

Enables validation of input XML files and generates post-schema-validation info for them. A value of `true` enables XML validation and generates post-schema-validation info for the XML files; `false` disables validation. Default value is `true`.

COM and .NET

Signature: `LoadXMLWithPSVI(boolean enable)`

Java

Signature: `public void setLoadXMLWithPSVI(boolean enable)`

6.4.1.5.2.13 MainOutput

Returns the main output of the last executed job.

COM and .NET

Signature: `string MainOutput()`

 © 2017-2023 Altova GmbH

Java

Signature: `public string getMainOutput()`

6.4.1.5.2.14 OutputEncoding

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.

COM and .NET

Signature: `OutputEncoding(string encoding)`

Java

Signature: `public void setOutputEncoding(string encoding)`

6.4.1.5.2.15 OutputIndent

Enables or disables indentation in the output document. A value of `true` enables indentation; `false` disables it.

COM and .NET

Signature: `OutputIndent(boolean outputIndent)`

Java

Signature: `public void setOutputIndent(boolean outputIndent)`

6.4.1.5.2.16 OutputMethod

Specifies the serialization of the output document. Valid values are: `xml|xhtml|html|text`. Default value is `xml`.

COM and .NET

Signature: `OutputMethod(string format)`

Java

Signature: `public void setOutputMethod(string format)`

6.4.1.5.2.17 OutputOmitXMLDeclaraton

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`.

COM and .NET

Signature: `OutputOmitXMLDeclaration(boolean omitDeclaration)`

Java

Signature: `public void setOutputOmitXMLDeclaration(boolean omitDeclaration)`

6.4.1.5.2.18 UpdatedXMLWriteMode

Specifies how updates to the XML file are handled. The property value is an ENUMXQueryUpdatedXML[345] literal.

COM and .NET

Signature: `UpdateXMLWriteMode(ENUMXQueryUpdatedXML[345] updateMode)`

Java

Signature: `public void setUpdateXMLWriteMode(ENUMXQueryUpdatedXML[345] updateMode)`

6.4.1.5.2.19 XincludeSupport

Enables or disables the use of `XInclude` elements. A value of `true` enables XInclude support; `false` disables it. The default value is `false`.

COM and .NET

Signature: `XincludeSupport(boolean xinclude)`

 © 2017-2023 Altova GmbH

Java

Signature: `public void setXincludeSupport(boolean xinclude)`

6.4.1.5.2.20 XMLValidationErrorsAsWarnings

Enables the treating of XML validation errors as warnings. Takes boolean `true` or `false`.

COM and .NET

Signature: `XMLValidationErrorsAsWarnings(boolean enable)`

Java

Signature: `public void setXMLValidationErrorsAsWarnings(boolean enable)`

6.4.1.5.2.21 XMLValidationMode

Sets the XML validation mode, which is an enumeration literal of ENUMXMLValidationMode[344] that determines whether to check validity or well-formedness.

COM and .NET

Signature: `XMLValidationMode(ENUMXMLValidationMode[344] valMode)`

Java

Signature: `public void setXMLValidationMode(ENUMXMLValidationMode[344] valMode)`

6.4.1.5.2.22 XQueryFileName

Specifies the XQuery file to use. The supplied string must be an absolute URL that gives the location of the XQuery file to use.

COM and .NET

Signature: `XQueryFileName(string fileurl)`

Java

Signature: `public void setXQueryFileName(string fileurl)`

6.4.1.5.2.23 XQueryFromText

Supplies, as a text string, the contents of the XQuery document to use

COM and .NET

Signature: `XQueryFromText(string xqtext)`

Java

Signature: `public void setXQueryFromText(string xqtext)`

6.4.1.5.2.24 XQueryUpdateVersion

Specifies the XQuery Update version to use. The property value is an ENUMXQueryVersion[346] literal.

COM and .NET

Signature: `XQueryUpdateVersion(ENUMXQueryVersion[346] version)`

Java

Signature: `public void setXQueryUpdateVersion(ENUMXQueryVersion[346] version)`

6.4.1.5.2.25 XSDVersion

Sets the XML Schema version against which the XML document will be validated. Value is an enumeration literal of ENUMXSDVersion[347].

COM and .NET

Signature: `XSDVersion(ENUMXSDVersion[347] version)`

 © 2017-2023 Altova GmbH

Java

Signature: `public void setXSDVersion(ENUMXSDVersion 347 version)`

6.4.1.6 XSLT

The **IXSLT/XSLT** interface/class provides methods to execute XSLT transformations and validate XSLT-related documents. You can also provide data for the transformation via external parameters.

Note that the name of the interface in the COM/.NET API is different than that of the class in the Java API:

- In COM/.NET: **IXSLT**
- In Java: **XSLT**
-

6.4.1.6.1 Methods

The methods of the **IXSLT** interface (COM/.NET) and **XSLT** class (Java) are described in this section.

6.4.1.6.1.1 AddExternalParameter

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. The `name` parameter holds the name of the variable, which is a QName, as a string. The `value` parameter holds the value of the variable as a string.

COM and .NET

Signature: `AddExternalParameter(string name, string value)`

Java

Signature: `public void addExternalParameter(string name, string value)`

6.4.1.6.1.2 ClearExternalParameterList

Clears the external parameters list created by the AddExternalParameter[327] method.

COM and .NET

Signature: **ClearExternalParameterList()**

Java

Signature: **public** void **clearExternalParameterList()**

6.4.1.6.1.3 Execute

Executes the XSLT transformation according to the XSLT specification named in the EngineVersion[331] property, and saves the result to the output file named in the outputFile parameter. If an error occurs, a RaptorXMLException[295] is raised. Use the LastErrorMessage property to access additional information.

COM and .NET

Signature: boolean **Execute(**string **outputFile)**

Java

Signature: **public** boolean **execute(**string **outputFile)**

6.4.1.6.1.4 ExecuteAndGetResultAsString

Executes the XSLT transformation according to the XSLT specification named in the EngineVersion[331] property, and returns the result as a string. This method does not produce additional result files, such as charts or secondary results. It also does not hold binary results such as .docx OOXML files. If additional output files are needed, use the Execute[328] method. If an error occurs, a RaptorXMLException[295] is raised. Use the LastErrorMessage property to access additional information.

COM and .NET

Signature: string **ExecuteAndGetResultAsString()**

Java

Signature: **public** string **executeAndGetResultAsString()**

 © 2017-2023 Altova GmbH

6.4.1.6.1.5 ExecuteAndGetResultAsStringWithBaseOutputURI

Executes the XSLT transformation according to the XSLT specification named in the EngineVersion[331] property, and returns the result as a string at the location defined by the base URI. The baseURI parameter is a string that provides a URI. This method does not produce additional result files, such as charts or secondary results. It also does not hold binary results such as .docx OOXML files. If additional output files are needed, use the Execute[328] method. If an error occurs, a RaptorXMLException[295] is raised. Use the LastErrorMessage property to access additional information.

COM and .NET

Signature: `string ExecuteAndGetResultAsStringWithBaseOutputURI(string baseURI)`

Java

Signature: `public string ExecuteAndGetResultAsStringWithBaseOutputURI(string baseURI)`

6.4.1.6.1.6 IsValid

Returns the result of validating the XSLT document according to the XSLT specification named in the EngineVersion[331] property. The result is `true` on success, `false` on failure. If an error occurs, a RaptorXMLException[295] is raised. Use the `LastErrorMessage` property to access additional information.

COM and .NET

Signature: `boolean IsValid()`

Java

Signature: `public boolean isValid()`

6.4.1.6.2 Properties

The properties of the **IXSLT** interface (COM/.NET) and **XSLT** class (Java) are described in this section.

6.4.1.6.2.1 AdditionalOutputs

Returns the additional outputs of the last executed job.

COM and .NET

Signature: `string AdditionalOutputs()`

Java

Signature: `public string getAdditionalOutputs()`

6.4.1.6.2.2 ChartExtensionsEnabled

Enables or disables Altova's chart extension functions. A value of `true` enables chart extensions; `false` disables them. Default value is `true`.

COM and .NET

Signature: `ChartExtensionsEnabled(boolean enable)`

Java

Signature: `public void setChartExtensionsEnabled(boolean enable)`

6.4.1.6.2.3 DotNetExtensionsEnabled

Enables or disables .NET extension functions. A value of `true` enables .NET extensions; `false` disables them. Default value is `true`.

COM and .NET

Signature: `DotNetExtensionsEnabled(boolean enable)`

Java

Signature: `public void setDotNetExtensionsEnabled(boolean enable)`

© 2017-2023 Altova GmbH

6.4.1.6.2.4 EngineVersion

Specifies the XSLT version to use. The property value is an ENUMXSLTVersion[348] literal.

COM and .NET

Signature: **EngineVersion(**ENUMXSLTVersion[348] **version)**

Java

Signature: **public** void **setEngineVersion(**ENUMXSLTVersion[348] **version)**

6.4.1.6.2.5 IndentCharacters

Submits the character string that will be used as indentation in the output.

COM and .NET

Signature: **IndentCharacters(**string **indentChars)**

Java

Signature: **public** void **setIndentCharacters(**string **indentChars)**

6.4.1.6.2.6 InitialTemplateMode

Sets the initial mode for XSLT processing. Templates with a mode value equal to the submitted string will be processed.

COM and .NET

Signature: **InitialTemplateMode(**string **mode)**

Java

Signature: **public** void **setInitialTemplateMode(**string **mode)**

6.4.1.6.2.7 InputXMLFileName

Sets the location, as a URL, of the XML document to process. The supplied string must be an absolute URL that gives the exact location of the XML file.

COM and .NET

Signature: `InputXMLFileName(string url)`

Java

Signature: `public void setInputXMLFileName(string url)`

6.4.1.6.2.8 InputXMLFromText

Supplies a string that is the text content of the XML document to process.

COM and .NET

Signature: `InputXMLFromText(string xml)`

Java

Signature: `public void setInputXMLFromText(string xml)`

6.4.1.6.2.9 JavaBarcodeExtensionLocation

Specifies the location of the barcode extension file. See the section on Altova's barcode extension functions 482 for more information. The supplied string must be an absolute URL that gives the base location of the file to use.

COM and .NET

Signature: `JavaBarcodeExtensionLocation(string url)`

Java

Signature: `public void setJavaBarcodeExtensionLocation(string url)`

 © 2017-2023 Altova GmbH

6.4.1.6.2.10 JavaExtensionsEnabled

Enables or disables Java extension functions. A value of `true` enables Java extensions; `false` disables them. Default value is `true`.

COM and .NET

Signature: **`JavaExtensionsEnabled(boolean enable)`**

Java

Signature: **`public void setJavaExtensionsEnabled(boolean enable)`**

6.4.1.6.2.11 LastErrorMessage

Retrieves a string that is the last error message from the RaptorXML engine.

COM and .NET

Signature: **`string LastErrorMessage()`**

Java

Signature: **`public string getLastErrorMessage()`**

6.4.1.6.2.12 LoadXMLWithPSVI

Enables validation of input XML files and generates post-schema-validation info for them. A value of `true` enables XML validation and generates post-schema-validation info for the XML files; `false` disables validation. Default value is `true`.

COM and .NET

Signature: **`LoadXMLWithPSVI(boolean enable)`**

Java

Signature: **`public void setLoadXMLWithPSVI(boolean enable)`**

6.4.1.6.2.13 MainOutput

Returns the main output of the last executed job.

COM and .NET

Signature: `string MainOutput()`

Java

Signature: `public string getMainOutput()`

6.4.1.6.2.14 NamedTemplateEntryPoint

Specifies the name, as a string, of the named template to use as an entry point for the transformation.

COM and .NET

Signature: `NamedTemplateEntryPoint(string template)`

Java

Signature: `public void setNamedTemplateEntryPoint(string template)`

6.4.1.6.2.15 SchemaImports

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 submitted.

COM and .NET

Signature: `SchemaImports(ENUMSchemaImports 340 importOption)`

Java

Signature: `public void setSchemaImports(ENUMSchemaImports 340 importOption)`

 © 2017-2023 Altova GmbH

6.4.1.6.2.16 SchemalocationHints

Specifies the mechanism to use to locate the schema. The mechanism is specified by the `ENUMLoadSchemalocation` literal that is selected.

COM and .NET

Signature: **SchemalocationHints(**ENUMLoadSchemalocation 339 **hint)**

Java

Signature: **public** void **setSchemalocationHints(**ENUMLoadSchemalocation 339 **hint)**

6.4.1.6.2.17 SchemaMapping

Sets what mapping to use in order to locate the schema. The mapping is specified by the `ENUMSchemaMapping` literal that is selected.

COM and .NET

Signature: **SchemaMapping(**ENUMSchemaMapping 342 **mappingOption)**

Java

Signature: **public** void **setSchemaMapping(**ENUMSchemaMapping 342 **mappingOption)**

6.4.1.6.2.18 StreamingSerialization

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.

COM and .NET

Signature: **StreamingSerialization(**boolean **enable)**

Java

Signature: **public** void **setStreamingSerialization(**boolean **enable)**

6.4.1.6.2.19 XincludeSupport

Enables or disables the use of `XInclude` elements. A value of `true` enables XInclude support; `false` disables it. The default value is `false`.

COM and .NET

Signature: **`XincludeSupport(`**`boolean` **`xinclude)`**

Java

Signature: **`public`** `void` **`setXincludeSupport(`**`boolean` **`xinclude)`**

6.4.1.6.2.20 XMLValidationErrorsAsWarnings

Enables the treating of XML validation errors as warnings. Takes boolean `true` or `false`.

COM and .NET

Signature: **`XMLValidationErrorsAsWarnings(`**`boolean` **`enable)`**

Java

Signature: **`public`** `void` **`setXMLValidationErrorsAsWarnings(`**`boolean` **`enable)`**

6.4.1.6.2.21 XMLValidationMode

Sets the XML validation mode, which is an enumeration literal of ENUMXMLValidationMode[344] that determines whether to check validity or well-formedness.

COM and .NET

Signature: **`XMLValidationMode(`**ENUMXMLValidationMode[344] **`valMode)`**

Java

Signature: **`public`** `void` **`setXMLValidationMode(`**ENUMXMLValidationMode[344] **`valMode)`**

© 2017-2023 Altova GmbH

6.4.1.6.2.22 XSDVersion

Sets the XML Schema version against which the XML document will be validated. Value is an enumeration literal of ENUMXSDVersion[347].

COM and .NET

Signature: **XSDVersion(**ENUMXSDVersion[347] **version)**

Java

Signature: **public** void **setXSDVersion(**ENUMXSDVersion[347] **version)**

6.4.1.6.2.23 XSLFileName

Specifies the XSLT file to use. The supplied string must be an absolute URL that gives the location of the XSLT file to use.

COM and .NET

Signature: **XSLFileName(**string **fileurl)**

Java

Signature: **public** void **setXSLFileName(**string **fileurl)**

6.4.1.6.2.24 XSLFromText

Supplies, as a text string, the contents of the XSLT document to use

COM and .NET

Signature: **XSLFromText(**string **xsltext)**

Java

Signature: **public** void **setXSLFromText(**string **xsltext)**

6.4.2 Enumerations

Enumerations of the COM/.NET and Java Server APIs are described in this section. Each description includes links to the methods or properties that use the enumeration.

- ENUMAssessmentMode 338
- ENUMErrorFormat 339
- ENUMLoadSchemalocation 339
- ENUMSchemaImports 340
- ENUMSchemaMapping 342
- ENUMValidationType 342
- ENUMWellformedCheckType 344
- ENUMXMLValidationMode 344
- ENUMXQueryUpdatedXML 345
- ENUMXQueryVersion 346
- ENUMXSDVersion 347
- ENUMXSLTVersion 348

6.4.2.1 ENUMAssessmentMode

Defines the assessment mode of the XML Validator to be strict or lax:

- **eAssessmentModeStrict**: Sets the schema-validity assessment mode to `Strict`. This is the default value.
- **eAssessmentModeLax**: Sets the schema-validity assessment mode to `Lax`.

COM and .NET

eAssessmentModeStrict	= 0
eAssessmentModeLax	= 1

Used by

Interface	*Property*
IXMLValidator 303	AssessmentMode 305

Java

```
public enum ENUMAssessmentMode {
       eAssessmentModeLax
       eAssessmentModeStrict   }
```

Used by

Class	*Method*

© 2017-2023 Altova GmbH

XMLValidator 303	setAssessmentMode 305

6.4.2.2 ENUMErrorFormat

Specifies the format of the error output:

- **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.

COM and .NET

eFormatText	= 0
eFormatShortXML	= 1
eFormatLongXML	= 2

Used by

Interface	*Property*
IServer 286	ErrorFormat 290

Java

```
public enum ENUMErrorFormat {
       eFormatText
       eFormatShortXML
       eFormatLongXML }
```

Used by

Class	*Method*
RaptorXMLFactory 286	setErrorFormat 290

6.4.2.3 ENUMLoadSchemalocation

Indicates how the schema's location should be determined. The selection is based on the schema location attribute of the XML instance document. This attribute could be `xsi:schemaLocation` or `xsi:noNamespaceSchemaLocation`.

- **eSHLoadBySchemalocation** uses the URL of the schema location attribute in the XML instance document. This enumeration literal is the **default value**.

- **eSHLoadByNamespace** 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.
- **eSHLoadCombiningBoth**: 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[342] decides which mapping is used. If neither the namespace nor schema location has a catalog mapping, the schema location URL is used.
- **eSHLoadIgnore**: The `xsi:schemaLocation` and `xsi:noNamespaceSchemaLocation` attributes are both ignored.

COM and .NET

eSHLoadBySchemalocation	= 0
eSHLoadByNamespace	= 1
eSHLoadCombiningBoth	= 2
eSHLoadIgnore	= 3

Used by

Interface	*Property*
IXMLValidator[303]	SchemalocationHints[313]
IXSLT[327]	SchemalocationHints[335]

Java

```
public enum ENUMLoadSchemalocation {
        eSHLoadBySchemalocation
        eSHLoadByNamespace
        eSHLoadCombiningBoth
        eSHLoadIgnore    }
```

Used by

Class	*Method*
XMLValidator[303]	setSchemalocationHints[313]
XSLT[327]	setSchemalocationHints[335]

6.4.2.4 ENUMSchemaImports

Defines the behavior of the schema's `xs:import` elements, each of which has an optional `namespace` attribute and an optional `schemaLocation` attribute.

 © 2017-2023 Altova GmbH

- **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 342 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.

COM and .NET

eSILoadBySchemalocation	= 0
eSILoadPreferringSchemalocation	= 1
eSILoadByNamespace	= 2
eSICombiningBoth	= 3
eSILicenseNamespaceOnly	= 4

Used by

Interface	*Property*
IXMLValidator 303	SchemaImports 312
IXSLT 327	SchemaImports 334

Java

```
public enum ENUMSchemaImports {
        eSILoadBySchemalocation
        eSILoadPreferringSchemalocation
        eSILoadByNamespace
        eSILoadCombiningBoth
        eSILicenseNamespaceOnly    }
```

Used by

Class	*Method*
XMLValidator 303	setSchemaImports 312
XSLT 327	setSchemaImports 334

6.4.2.5 ENUMSchemaMapping

Specifies which of two catalog mappings is preferred: namespaces or schema-location URLs. This enumeration is useful for disambiguating ENUMLoadSchemalocation[339] and ENUMSchemaImports[340].

- **eSMPreferNamespace**: Selects the namespace.
- **eSMPreferSchemalocation**: Selects the schema location. This is the default value.

COM and .NET

eSMPreferSchemalocation	= 0
eSMPreferNamespace	= 1

Used by

Interface	*Property*
IXMLValidator[303]	SchemaMapping[313]
IXSLT[327]	SchemaMapping[335]

Java

```
public enum ENUMSchemaMapping {
       eSMPreferSchemalocation
       eSMPreferNamespace    }
```

Used by

Class	*Method*
IXMLValidator[303]	setSchemaMapping[313]
IXSLT[327]	setSchemaMapping[335]

6.4.2.6 ENUMValidationType

Specifies what validation to carry out and, in the case of XML documents, whether validation is against a DTD or XSD.

- **eValidateAny**: The document type (for example, XML or XSD) is detected, and validation is set automatically for that document type.
- **eValidateXMLWithDTD**: Specifies validation of an XML document against a DTD.
- **eValidateXMLWithXSD**: Specifies validation of an XML document against an XSD (XML Schema).
- **eValidateDTD**: Specifies validation of a DTD document.

 © 2017-2023 Altova GmbH

- **eValidateXSD**: Specifies validation of an XSD (W3C XMLSchema) document.
- **eValidateJSON**: Specifies validation of a JSON instance document.
- **eValidateJSONSchema**: Specifies validation of a JSON Schema document according to JSON Schema v4.
- **eValidateAvro**: Specifies validation of an Avro binary file. The Avro data in the binary file is validated against the Avro Schema contained in the binary file.
- **eValidateAvroSchema**: Specifies validation of an Avro schema against the Avro schema specification.
- **eValidateAvroJSON**: Specifies validation of a JSON-serialized Avro data file against an Avro schema.

COM and .NET

eValidateAny	= 0
eValidateXMLWithDTD	= 1
eValidateXMLWithXSD	= 2
eValidateDTD	= 3
eValidateXSD	= 4
eValidateJSON	= 5
eValidateJSONSchema	= 6
eValidateAvro	= 7
eValidateAvroSchema	= 8
eValidateAvroJSON	= 9

Used by

Interface	*Method*
IXMLValidator 303	isValid 304

Java

```
public enum ENUMValidationType {
        eValidateAny
        eValidateXMLWithDTD
        eValidateXMLWithXSD
        eValidateDTD
        eValidateXSD
        eValidateJSON
        eValidateJSONSchema
        eValidateAvro
        eValidateAvroSchema
        eValidateAvroJSON }
```

Used by

Class	*Method*
XMLValidator 303	isValid 304

6.4.2.7 ENUMWellformedCheckType

Specifies the type of well-formed check to make (for XML, DTD, or JSON).

- **eWellformedAny**: The document type is detected, and the type of check is set automatically.
- **eWellformedXML**: Checks an XML document for well-formedness.
- **eWellformedDTD**: Checks a DTD document for well-formedness.
- **eWellformedJSON**: Checks a JSON document for well-formedness.

COM and .NET

eWellFormedAny	= 0
eWellFormedXML	= 1
eWellFormedDTD	= 2
eWellFormedJSON	= 3

Used by

Interface	*Method*
IXMLValidator 303	isWellFormed 305

Java

```
public enum ENUMWellformedCheckType {
       eWellformedAny
       eWellformedXML
       eWellformedDTD
       eWellformedJSON    }
```

Used by

Class	*Method*
XMLValidator 303	isWellFormed 305

6.4.2.8 ENUMXMLValidationMode

Specifies the type of XML validation to perform (validation or well-formedness check).

- **eProcessingModeWF**: Sets the XML processing mode to `wellformed`. This is the default value.
- **eProcessingModeValid**: Sets the XML processing mode to `validation`.
- **eProcessingModeID**: Internal, not for use.

 © 2017-2023 Altova GmbH

COM and .NET

eXMLValidationModeWF	= 0
eXMLValidationModeID	= 1
eXMLValidationModeValid	= 2

Used by

Interface	*Property*
IXMLValidator 303	XMLValidationMode 314
IXQuery 315	XMLValidationMode 325
IXSLT 327	XMLValidationMode 336

Java

```
public enum ENUMXMLValidationMode {
       eProcessingModeValid
       eProcessingModeWF
       eProcessingModeID  }
```

Used by

Class	*Method*
XMLValidator 303	setXMLValidationMode 314
XQuery 315	setXMLValidationMode 325
XSLT 327	setXMLValidationMode 336

6.4.2.9 ENUMXQueryUpdatedXML

Specifies how XQuery updates are handled.

- **eUpdatedDiscard**: Updates are discarded and not written to file.
- **eUpdatedWriteback**: Updates are written to the input XML file specified with (set) InputXMLFileName 320.
- **eUpdatedAsMainResult**: Updates are written to the location specified by the `outputFile` parameter of ExecuteUpdate 297.

COM and .NET

eUpdatedDiscard	= 1
eUpdatedWriteback	= 2

eUpdatedAsMainResult	= 3

Used by

Interface	*Property*
IXQuery 315	UpdatedXMLWriteMode 324

Java

```
public enum ENUMXQueryUpdatedXML {
        eUpdatedDiscard
        eUpdatedWriteback
        eeUpdatedAsMainResult }
```

Used by

Class	*Method*
XQuery 315	setUpdatedXMLWriteMode 324

6.4.2.10 ENUMXQueryVersion

Sets the XQuery version to be used for processing (execution or validation).

- **eXQVersion10**: Sets the XQuery version to XQuery 1.0.
- **eXQVersion30**: Sets the XQuery version to XQuery 3.0. The default value.
- **eXQVersion31**: Sets the XQuery version to XQuery 3.1.

Note: The Java enumeration literals are differently named than the COM/.NET literals. *See below.*

COM and .NET

eXQVersion10	= 1
eXQVersion30	= 3
eXQVersion31	= 31

Used by

Interface	*Property*
IXQuery 315	EngineVersion 319

Java

```
public enum ENUMXQueryVersion {
        eVersion10
        eVersion30
```

 © 2017-2023 Altova GmbH

```
        eVersion31 }
```

Used by

Class	Method
XQuery 315	setEngineVersion 319

6.4.2.11 ENUMXSDVersion

Specifies the XML Schema version to use for validation.

- **eXSDVersionAuto**: The XML Schema version is detected automatically from the XSD document's `vc:minVersion` attribute. If this attribute's value is `1.1`, then the document is considered to be XSD 1.1. If the attribute has any other value, or if no value exists, then the document is 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.

COM and .NET

eXSDVersionAuto	= 0
eXSDVersion10	= 1
eXSDVersion11	= 2

Used by

Interface	Property
IXMLValidator 303	XSDVersion 315
IXQuery 315	XSDVersion 326
IXSLT 327	XSDVersion 337

Java

```
public enum ENUMXSDVersion {
        eXSDVersionAuto
        eXSDVersion10
        eXSDVersion11    }
```

Used by

Class	Method
XMLValidator 303	setXSDVersion 315
XQuery 315	setXSDVersion 326
XSLT 327	setXSDVersion 337

6.4.2.12 ENUMXSLTVersion

Sets the XSLT version to be used for processing (validation or XSLT transformation).

- **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.

COM and .NET

eVersion10	= 1
eVersion20	= 2
eVersion30	= 3

Used by

Interface	*Property*
IXSLT 327	EngineVersion 331

Java

```
public enum ENUMXSLTVersion {
       eVersion10
       eVersion20
       eVersion30 }
```

Used by

Class	*Method*
XSLT 327	setEngineVersion 331

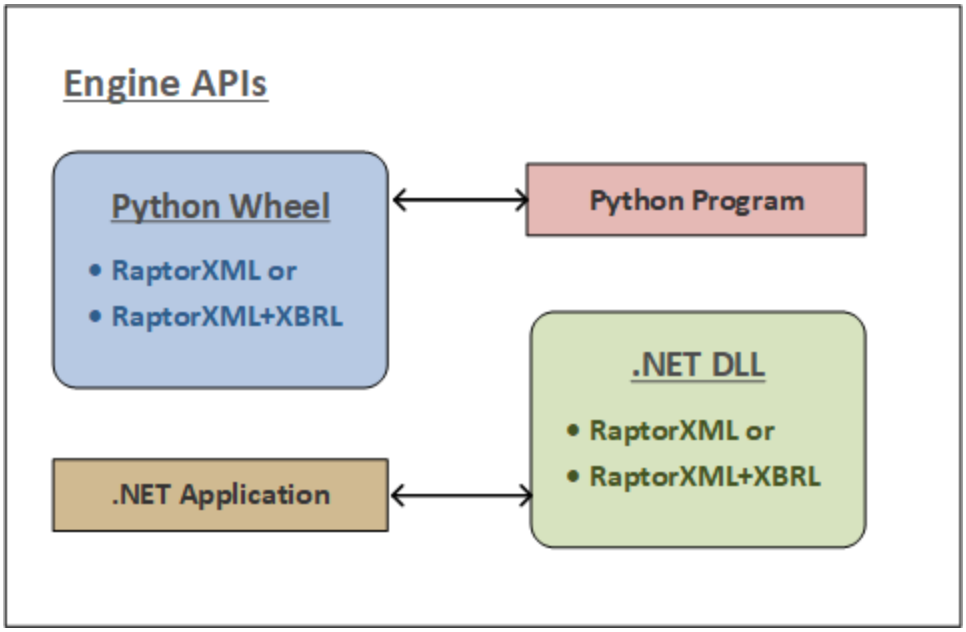
© 2017-2023 Altova GmbH

7 Engine APIs: Python and .NET

RaptorXML Server provides two engine APIs:

- a Python wheel file (`.whl`), which is the Python Engine API: `raptorxml<versiondetails>.whl`
- a .NET DLL file (`.dll`), which is the .NET Engine API: `raptorxmlapi.dll`

These two engine APIs provide the RaptorXML Server functionality as separate packages that are standalone and independent from RaptorXML Server (*see figure below*). Each package must be installed on the user's machine before it can be imported as a Python module or integrated into a custom .NET application. Because all processing is performed locally on the user's machine, the Python and .NET engine APIs provide detailed access to the data models of any valid XML and XBRL instances, XSD schemas and XBRL taxonomies. The APIs expose a rich set of methods to iterate over the content of XBRL instances or allow to retrieve specific bits of information from XBRL taxonomies with a few lines of code.



Note the following points about the Engine APIs:

- After you install RaptorXML Server, both engine APIs will be located in the `bin` folder of the RaptorXML Server installation folder.
- The engine APIs provide additional advanced processing via more versatile objects in their APIs.
- In order to use an engine API, a licensed version of RaptorXML Server must be installed on the machine on which the Python program or .NET application is executed (see *Usage* below).

Usage

You can create a Python program or .NET application as follows:

Python program

A Python program can access RaptorXML functionality by using Python API objects 352 (see here 352). When the Python program is executed, it will use the RaptorXML library that has been installed in your Python

environment when you install the Python wheel. Note that the Python wheel is compatible with Python version 3.11.5 **only**.

.NET application
A .NET application can access RaptorXML functionality by using .NET API objects 362 (see here 362). When the .NET application is executed, it will use the RaptorXML that is contained in the .NET API DLL.

Licensing

In order to use an engine API, a licensed version of RaptorXML Server must be installed on the machine on which the Python program or .NET application is executed. See the section Licensing 351 for more detailed information.

© 2017-2023 Altova GmbH

7.1 Licensing

In order for an API package to run on a client machine, that machine will have to be licensed as a RaptorXML Server client. Licensing consists of two steps:

1. Registering the machine as a RaptorXML Server client with Altova LicenseServer
2. Assigning a RaptorXML Server license from LicenseServer to that machine.

If you plan to use the API package from a given machine, then two possible situations arise:

- If the client machine is already running a licensed installation of RaptorXML Server, then the API package can be run without you needing to take any additional steps. This is because the machine is already licensed to run RaptorXML Server. Consequently, use of the API package on this machine is covered by the license assigned to RaptorXML Server on that machine.
- If RaptorXML Server is not installed on the client machine and you do not want to install RaptorXML Server on that machine for whatever reason. In this case, you can still register the machine as a RaptorXML Server client and assign it a RaptorXML Server license. How to do this is described below.

To register a machine (on which RaptorXML Server is not installed) as a RaptorXML Server client, use the command line application **registerlicense.exe**, which is located in the application's **bin** folder:

Windows	`Program Files\Altova\RaptorXMLServer2024\bin`
Linux	`/opt/Altova/RaptorXMLServer2024/bin`
Mac	`/usr/local/Altova/RaptorXMLServer2024/bin`

On the command line run the command:

```
registerlicense <LicenseServer>
```

where **<LicenseServer>** is the IP address or host name of the LicenseServer machine.

This command will register the machine as a RaptorXML Server client with Altova LicenseServer. For information about how to assign a RaptorXML Server license to the machine and for more information about licensing, see the Altova LicenseServer documentation.

Deploying on Linux

To deploy the **registerlicense** application with your Python wheel package, the shared libraries that are listed below need to be present in a sibling **lib** directory. The shared libraries can be copied from your Raptor installation folder:

```
/opt/Altova/RaptorXMLServerRaptorXMLServer2024/lib
```

- libcrypto.so.1.0.0
- libssl.so.1.0.0
- libstdc++.so.6
- libtbb.so.2

7.2 Python API

The RaptorXML Python API enables data in XML documents and XML Schema documents to be accessed and manipulated in Python scripts. Some typical use cases of the Python API include:

- implement custom validation rules and error messages
- export content from XML documents to a database
- export content from XML documents to custom data formats
- interactively navigate and query the data model of XML documents within a Python shell or Jupyter notebook (http://jupyter.org/)

The Python APIs

The Python APIs (for XML and XSD) provide access to the meta-information, structural information, and data contained in XML and XSD 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 Server that writes data from an XML document to a database or to a CSV file.

Example scripts for Raptor's Python APIs are available at: https://github.com/altova

The Python APIs are described in their API references:

- Python API v1 Reference
- Python API v2 Reference

Note: Raptor's **Python API v1 is deprecated**. Please use Python API v2.

RaptorXML Server package for Python

In your installation of RaptorXML Server, you will also find a Python package in wheel format. You can use Python's `pip` command to install this package as a module of your Python installation. After the RaptorXML module has been installed, you can use the module's functions within your code. In this way, RaptorXML's functionality can be used easily in any Python program you write, together with other third-party Python libraries, such as graphics libraries.

For information about how to use RaptorXML Server's Python package, see the section RaptorXML Server as a Python Package 355.

Note: The Python wheel in v2024 and later is compatible with Python version 3.11.5 and higher.

Python scripts

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

- valxml-withxsd (xsi) 62
- valxsd (xsd) 73

These commands invoking Python scripts can be used both on the Command Line Interface (CLI) 56 and via the HTTP Interface 238. The usage of Python scripts with the Python APIs of RaptorXML Server are described at: https://github.com/altova.

 © 2017-2023 Altova GmbH

Making Python scripts safe

When a Python script is specified in a command via HTTP to RaptorXML Server, the script will only work if it is located in the trusted directory[243]. 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`[243] setting of the server configuration file[242], 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[243] (which is a sub-directory of the `output-root-directory`[243]), 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[243] for potential vulnerability issues.

7.2.1 Python API Versions

RaptorXML Server supports multiple Python API versions. Any previous Python API version is also supported by the current version of RaptorXML Server. The Python API version is selected by the **`--script-api-version=MAJOR_VERSION`** command line flag. The default of the `MAJOR_VERSION` argument is always the current version. A new RaptorXML Server Python API `MAJOR_VERSION` is introduced when incompatible changes or major enhancements are introduced. Users of the API do not need to upgrade their existing scripts when a new major version is released.

It is recommended that:

- You use the `--script-api-version=MAJOR_VERSION` flag to invoke utility scripts from the RaptorXML Server command-line (or Web-API). This ensures that scripts still work as expected after RaptorXML Server updates—even if a new API `MAJOR_VERSION` has been released.
- You use the latest version of the API for new projects, even though previous versions will be supported by future RaptorXML Server releases.

The Python API versions listed below are currently available. The documentation of the different APIs are available online at the locations given below.

Example files

For examples of scripts that use Raptor's Python APIs, go to https://github.com/altova.

Python API version 1

Introduced with RaptorXML Server v2014

Command line flag: **`--script-api-version=1`**
Documentation: Python API Version 1 Reference

This is the original RaptorXML Server Python API. It covers support to access the internal model of RaptorXML Server for:

- XML 1.0 and XML 1.1 (API module **xml**)
- XMLSchema 1.0 and XMLSchema 1.1 (API module **xsd**)
- XBRL 2.1 (API module **xbrl**)

The API can be used through callback functions which are implemented in a Python script file.

- on_xsi_valid
- on_xsd_valid
- on_dts_valid
- on_xbrl_valid

A script is specified with the `--script` option on the command line. The callback functions are invoked only if the validation succeeds. Details about the callback functions and the API are described in the RaptorXML Server Python API version 1 reference.

Note: Raptor's **Python API v1 is deprecated**. Please use Python API v2.

Python API version 2

Introduced with RaptorXML Server v2015r3. The latest API version is **2.8.6**.

Command line flag	Release
--script-api-version=2	*v 2015r3*
--script-api-version=2.1	*v 2015r4*
--script-api-version=2.2	*v 2016*
--script-api-version=2.3	*v 2016r2*
--script-api-version=2.4	*v 2017*
--script-api-version=2.4.1	*v 2018*
--script-api-version=2.5.0	*v 2018r2*
--script-api-version=2.6.0	*v 2019*
--script-api-version=2.7.0	*v2019r3*
--script-api-version=2.8.0	*v2020*
--script-api-version=2.8.1	*v2020r2*
--script-api-version=2.8.2	*v2021*
--script-api-version=2.8.3	*v2021r2*
--script-api-version=2.8.4	*v2022r2*
--script-api-version=2.8.5	*v2023r2sp1*
--script-api-version=2.8.6	*v2024*
Documentation: Python API Version 2 Reference	

© 2017-2023 Altova GmbH

This API version introduces over 300 new classes and reorganizes the modules from the RaptorXML Server Python API version 1 in such a way that frequently used information (for example, PSVI data) can be accessed more simply and related APIs are grouped logically together (fr example, `xbrl.taxonomy`, `xbrl.formula`, `xbrl.table`). In this version, the callback functions are invoked not only if validation succeeds, but also if validation fails. To reflect this behavior, the name of the callback functions are changed to:

- `on_xsi_finished`
- `on_xsd_finished`
- `on_dts_finished`
- `on_xbrl_finished`

To enable modularization, RaptorXML Server now supports multiple `--script` options. The callbacks implemented in these Python script files are executed in the order specified on the command line.

7.2.2 RaptorXML Server as a Python Package

Starting with RaptorXML Server 2024, the Python API is available as a native Python wheel package for **Python 3.11.5**. The Python wheel package can be installed as an extension module in your favored Python 3.11.5 distribution (for example, from python.org). Some Python 3 distributions (for example, from jupyter.org, anaconda.org and SciPy.org) include a wide range of extension modules for big data, mathematics, science, engineering and graphics. These modules now become available to RaptorXML Server without the need to build these modules specifically for RaptorXML Server. Otherwise, the wheel package works the same way as the **`RaptorXMLXBRL-python.exe`** application that is included with RaptorXML Server.

Note: The Python wheel package is a native Python 3.11.5 extension module and is compatible with Python version 3.11.5.

Note: The Python wheel package does not include the Python API v1.

Note: If you update your version of RaptorXML Server, make sure to update the Python wheel package in your Python environment.

The information required to correctly install the RaptorXML Server package is given in the sections below:

- Name of wheel file 356
- Location of wheel file 356
- Installing a wheel with pip 356
- Troubleshooting the installation 356
- The root catalog file 356
- The JSON config file 357

For information about how to use RaptorXML Server's Python API, see the Python API Reference and examples 353. Also see example scripts that use Raptor's Python API at https://github.com/altova.

Name of wheel file

Wheel files are named according to the following pattern:

```
raptorxmlserver-{version}(-{build tag})?-{python tag}-{abi tag}-{platform tag}.whl
```

Example:
raptorxmlserver-2.8.1-cp35-cp35m-win_amd64.whl

Location of wheel file

A wheel file is packaged with your installation of RaptorXML Server. It is located in the application's **bin** folder:

Windows	`Program Files\Altova\RaptorXMLServer2024\bin`
Linux	`/opt/Altova/RaptorXMLServer2024/bin`
Mac	`/usr/local/Altova/RaptorXMLServer2024/bin`

Installing a wheel with pip

To install the RaptorXML Server package as a Python module, use the **pip** command:

```
pip install <wheel-file>.whl
python -m pip install <wheel-file>.whl
```

If you have installed Python 3.11.5 or later from python.org, then `pip` will already be installed. Otherwise, you will need to install **pip** first. For more information, see https://docs.python.org/3/installing/.

Troubleshooting the installation

In case you are using older versions of the Python interpreter, you might have to adjust your installation to use the latest **vcruntime** libraries on windows or standard C++ libraries on Unix. These libraries are distributed with RaptorXML Server and can be used as described below.

Windows
If the **vcruntime140_1.dll** is missing, copy it from the `Program Files\Altova\RaptorXMLServer2024\bin` folder to the Python installation folder (the folder containing **python.exe**). (More generally, the Python interpreter needs to know where to find DLLs or shared libraries.)

Linux
If your system's C++ library is outdated, then your Python interpreter will not know how to find the newer C++ library that is used by the RaptorXML Server Python package and distributed with RaptorXML Server. This can be fixed by using **$LD_LIBRARY_PATH** to point to the newer library in the RaptorXML Server folder, like this: `$ export LD_LIBRARY_PATH=/opt/Altova/RaptorXMLServer2024/lib`.

macOS
If your system's C++ library is outdated, then your Python interpreter will not know how to find the newer C++ library that is used by the RaptorXML Server Python package and distributed with RaptorXML Server. This can be fixed by using **$DYLD_LIBRARY_PATH** to point to the newer library in the RaptorXML Server folder, like this: `$ export DYLD_LIBRARY_PATH=/usr/local/Altova/RaptorXMLServer2024/lib`.

The root catalog file

The RaptorXML module for Python must be able to locate **RootCatalog.xml**, the root catalog file that is stored in your RaptorXML Server installation folder. This is so that the RaptorXML module can use the catalog to correctly locate the various resources, such as schemas and other specifications, that the module references

 © 2017-2023 Altova GmbH

in order to carry out functions such as validations and transformations. The RaptorXML module will automatically locate **RootCatalog.xml** if the catalog's location has not been changed subsequent to the installation of RaptorXML Server.

In case you move or modify your RaptorXML Server environment, or if you move **RootCatalog.xml** from its original installed location, then you can specify the catalog's location by means of environment variables and the RaptorXML module's JSON Config File[357]. See the list below for the various ways in which you can do this. The RaptorXML module determines the location of **RootCatalog.xml** by looking up the following resources in the order given.

1	Environment variable `ALTOVA_RAPTORXML_PYTHON_CATALOGPATH`	Create with a value that is the path to **RootCatalog.xml**
2	HKLM Registry: `SOFTWARE\Altova\RaptorXMLServer\Installation_v2024_x64\Setup\CatalogPath`	Registry key is added by RaptorXML Server installer. Value is the path to **RootCatalog.xml**. *Windows only*
3	Location: `/opt/Altova/RaptorXMLServer2024/etc/RootCatalog.xml`	*Linux only*
4	Location: `/usr/local/Altova/RaptorXMLServer2024/etc/RootCatalog.xml`	*Mac only*
5	Environment variable `ALTOVA_RAPTORXML_PYTHON_CONFIG`	Create with a value that is the path to the JSON config file[357].
6	Location: `.altova/`**raptorxml-python.config**	The JSON config file[357] in the current working directory
7	Location: `~/.config/altova/`**raptorxml-python.config**	The JSON config file[357] in the user's home directory
8	Location: `/etc/altova/altova/`**raptorxml-python.config**	The JSON config file[357]. *Linux and Mac only*

The JSON config file

You can create a JSON config file for the RaptorXMLServer module. This file will be used by options 5 to 8 in the table above to locate the root catalog file[356]. The JSON config file must contain a map with a `"CatalogPath"` key that has a value which is the path to the root catalog file[356].

Listing of JSON config file

```
{
    "CatalogPath": "/path/to/RootCatalog.xml"
}
```

7.2.3 Debugging Server-Side Python Scripts

Most of the debugging functionality—apart the server-specific callbacks—can be used in a standard Python interpreter or (virtual) environment after the RaptorXML Server module has been installed by using `pip`:

```
pip install –upgrade "/path/to/RaptorXML/application-folder/bin/raptorxml-version-cp37-cp37m-winversion.whl"
```

After installing the wheel, you should be able to use any Python IDE to debug a script. You could try to extract the main functionality into a separate function which takes an instance object. This can then be called (i) by the RaptorXML Server callbacks, or (ii) by directly executing the script with a Python interpreter.

```
from altova_api.v2 import xml, xsd, xbrl

def main(instance):
    # Here goes the application specific logic

# Main entry point, will be called by RaptorXML after the XML instance validation job has finished
def on_xsi_finished(job, instance):
    # instance object will be None if XML Schema validation was not successful
    if instance:
        main(instance)

# Main entry point, will be called by RaptorXML after the XBRL instance validation job has finished
def on_xbrl_finished(job, instance):
    # instance object will be None if XBRL 2.1 validation was not successful
    if instance:
        main(instance)

if __name__ == '__main__':
    # parse arguments and create an instance
    instance = …
    main(instance)
```

7.2.4 Debugging Python Scripts in Visual Studio Code

We assume an up-to-date Visual Studio Code (VS Code) intallation with the `ms-python.python` extension installed. Please read the official Python debug configurations in Visual Studio Code guide for a general overview.

Note the following points:

- This guide uses `raptorxml-python` as the command to execute RaptorXML Server as a Python interpreter.
- The `raptorxml-python` executable is available in the `bin` folder of your RaptorXML Server application folder.

 © 2017-2023 Altova GmbH

Overview

We introduce two methods to use VS Code to debug Python scripts in RaptorXML Server.

- Method 1 also works for servers and RaptorXML Python callbacks (`--script` option).
- Method 2 doesn't require any source code modifications. It is a modified invocation of RaptorXML. Method 2 doesn't work for servers and RaptorXML Python callbacks (`--script` option).
- Both methods work with a standard Python interpreter and the imported RaptorXML Python module (`'import altova_api.v2 as altova'`).

Method 1: Change your source code

Carry out the following steps:

1. Run: `raptorxml-python -m pip install --upgrade debugpy`
2. Add the following lines to your Python source code:

```
python
import debugpy
debugpy.listen(5678)
debugpy.wait_for_client()
debugpy.breakpoint()
```

3. Copy this launch configuration to VS Code `launch.json` (defaults will do for the above values) and select it for `Run`.

```
json5
  {
    "name": "Python: Remote Attach",
    "type": "python",
    "request": "attach",
    "connect": {
    "host": "localhost",
    "port": 5678
    },
    "pathMappings": [
      {
        "localRoot": "${workspaceFolder}",
        "remoteRoot": "."
      }
    ]
  }
```

   You can also run by using the menu command **Run->Add Configuration...->Python->Remote Attach** with the defaults accepted.

4. Run your Python script (or RaptorXML with `--script` callbacks) as usual.
5. Start debugging (usually with the shortcut **F5**).

Method 2: Use a modified command line

Carry out the following steps:

1. Add a launch configuration (as in Method 1 above), and select it for `Run`.

2. Set a breakpoint in your Python script.
3. Run the command: `raptorxml-python -m debugpy --listen 0.0.0.0:5678 --wait-for-client your-script-.py`
4. Start debugging (usually with the shortcut **F5**).

Note: Debugging also works with containers and remote servers. You have to change the `host` key of the `connect` entry in the launch configuration. You may also use other ports as long as code or command line and `launch.json` have consistent values.

Setting raptorxml-python.exe as VS Code's default interpreter

It is possible to configure `raptorxml-python.exe` as the default Python interpreter of VS Code. Do this by adding the following to your VS Code `settigs.json` file:

```
json
  "python.defaultInterpreterPath": "/path/to/raptorxml-python.exe"
   ...
```

In this case, it is also possible to use a "Current File" launch configuration that starts the script for debugging. Consult the official VS Code documentation for details.

7.2.5 FAQs

Q: *I want to write a Python script that creates a new XML instance one element at a time while running inside the raptor server. These need to be serialized to the output with different encodings and formatting depending on parameters. Is this possible in RaptorXML Server.*

A: No, this is currently not possible because we do not have an API for creating arbitrary XML instances. However, when it comes to generating XBRL instances, we do have a high-level API which manages a lot of the technical details (such as avoiding writing duplicate contexts/units, and lots more). See https://www.altova.com/manual/en/raptorapi/pyapiv2/2.8.6/html/xbrl.InstanceDocumentBuilder.html for more information.

Q: *I would like to use lxml. Can I install lxml libraries into the Python folder at "RaptorXMLXBRLServer2024/lib/"?*

A: You can install most Python modules directly by running the following command in a terminal that has administrator rights:

```
"/path/to/RaptorXML/application-folder/bin/RaptorXMLXBRL-python.exe" -m pip install lxml
```

Q: *Would it be all right to create a big string that contains the XML instance, then parse the whole thing and re-serialize it.*

A: That is one possibility. You can parse and validate XML and XBRL instances from a string buffer using the Python API like this:

```
from altova_api.v2 import xml
txt = '''<?xml version="1.0" encoding="utf-8"?>
```

 © 2017-2023 Altova GmbH

```
<doc>
    <elem attr="foo">bar</elem>
</doc>'''
inst = xml.Instance.create_from_buffer(txt.encode('utf-8')).result
print(inst.serialize())
```

7.3 .NET Framework API

The **.NET Framework API** of RaptorXML Server enables you to integrate **the RaptorXML engine** in applications written in C# and other .NET languages.

It is implemented as a .NET assembly and puts **the RaptorXML engine** directly inside an application or a .NET-framework-based extension mechanism like VSTO (Visual Studio Tools for Office). The API provides fine-grained access to validate documents and to query their internal data model from RaptorXML Server.

Reference and resources

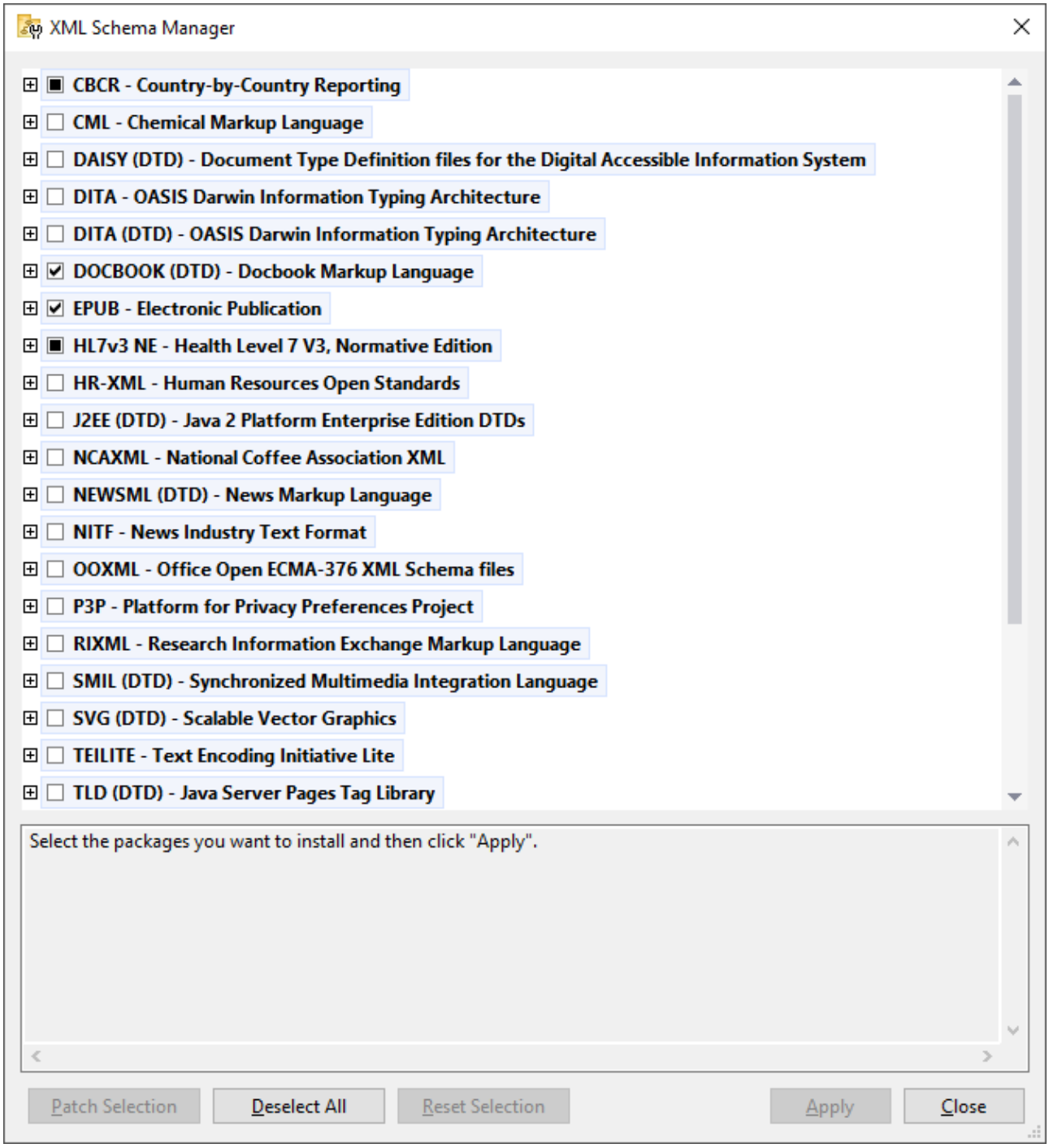
- *API documentation:* The latest RaptorXML Server .NET Framework API documentation is located at https://www.altova.com/manual/en/raptorapi/dotnetapiv2/2.8.6/html/index.html.
- *Example code:* The example code is hosted at https://github.com/altova/RaptorXML-Examples.

© 2017-2023 Altova GmbH

8 Schema Manager

XML Schema Manager is an Altova tool that provides a centralized way to install and manage XML schemas (DTDs for XML and XML Schemas) for use across all Altova's XML-Schema-aware applications, including RaptorXML Server.

- On Windows, Schema Manager has a graphical user interface (*screenshot below*) and is also available at the command line. (Altova's desktop applications are available on Windows only; *see list below.*)
- On Linux and macOS, Schema Manager is available at the command line only. (Altova's server applications are available on Windows, Linux, and macOS; *see list below.*)



Altova applications that operate with Schema Manager

Desktop applications (Windows only)	Server applications (Windows, Linux, macOS)
XMLSpy (all editions)	RaptorXML Server, RaptorXML+XBRL Server

© 2017-2023 Altova GmbH

MapForce (all editions)	StyleVision Server
StyleVision (all editions)	
Authentic Desktop Enterprise Edition	

Installation and de-installation of Schema Manager

Schema Manager is installed automatically when you first install a new version of Altova Mission Kit or of any of Altova's XML-schema-aware applications (*see table above*).

Likewise, it is removed automatically when you uninstall the last Altova XML-schema-aware application from your computer.

Schema Manager features

Schema Manager provides the following features:

- Shows XML schemas installed on your computer and checks whether new versions are available for download.
- Downloads newer versions of XML schemas independently of the Altova product release cycle. (Altova stores schemas online, and you can download them via Schema Manager.)
- Install or uninstall any of the multiple versions of a given schema (or all versions if necessary).
- An XML schema may have dependencies on other schemas. When you install or uninstall a particular schema, Schema Manager informs you about dependent schemas and will automatically install or remove them as well.
- Schema Manager uses the XML catalog mechanism to map schema references to local files. In the case of large XML schemas, processing will therefore be faster than if the schemas were at a remote location.
- All major schemas are available via Schema Manager and are regularly updated for the latest versions. This provides you with a convenient single resource for managing all your schemas and making them readily available to all of Altova's XML-schema-aware applications.
- Changes made in Schema Manager take effect for all Altova products installed on that machine.
- In an Altova product, if you attempt to validate on a schema that is not installed but which is available via Schema Manager, then installation is triggered automatically. However, if the schema package contains namespace mappings, then there will be no automatic installation; in this case, you must start Schema Manager, select the package/s you want to install, and run the installation. If, after installation, your open Altova application does not restart automatically, then you must restart it manually.

How it works

Altova stores all XML schemas used in Altova products online. This repository is updated when new versions of the schemas are released. Schema Manager displays information about the latest available schemas when invoked in both its GUI form as well as on the CLI. You can then install, upgrade or uninstall schemas via Schema Manager.

Schema Manager also installs schemas in one other way. At the Altova website (https://www.altova.com/schema-manager) you can select a schema and its dependent schemas that you want to install. The website will prepare a file of type `.altova_xmlschemas` for download that contains information about your schema selection. When you double-click this file or pass it to Schema Manager via the CLI as an argument of the `install` 377 command, Schema Manager will install the schemas you selected.

Local cache: tracking your schemas
All information about installed schemas is tracked in a centralized cache directory on your computer, located here:

Windows	`C:\ProgramData\Altova\pkgs\.cache`
Linux	`/var/opt/Altova/pkgs\.cache`
macOS	`/var/Altova/pkgs`

This cache directory is updated regularly with the latest status of schemas at Altova's online storage. These updates are carried out at the following times:

- Every time you start Schema Manager.
- When you start RaptorXML Server for the first time on a given calendar day.
- If RaptorXML Server is open for more than 24 hours, the cache is updated every 24 hours.
- You can also update the cache by running the `update` 380 command at the command line interface.

The cache therefore enables Schema Manager to continuously track your installed schemas against the schemas available online at the Altova website.

Do not modify the cache manually!

The local cache directory is maintained automatically based on the schemas you install and uninstall. It should not be altered or deleted manually. If you ever need to reset Schema Manager to its original "pristine" state, then, on the command line interface (CLI): (i) run the `reset` 378 command, and (ii) run the `initialize` 376 command. (Alternatively, run the **`reset`** command with the **`--i`** option.)

© 2017-2023 Altova GmbH

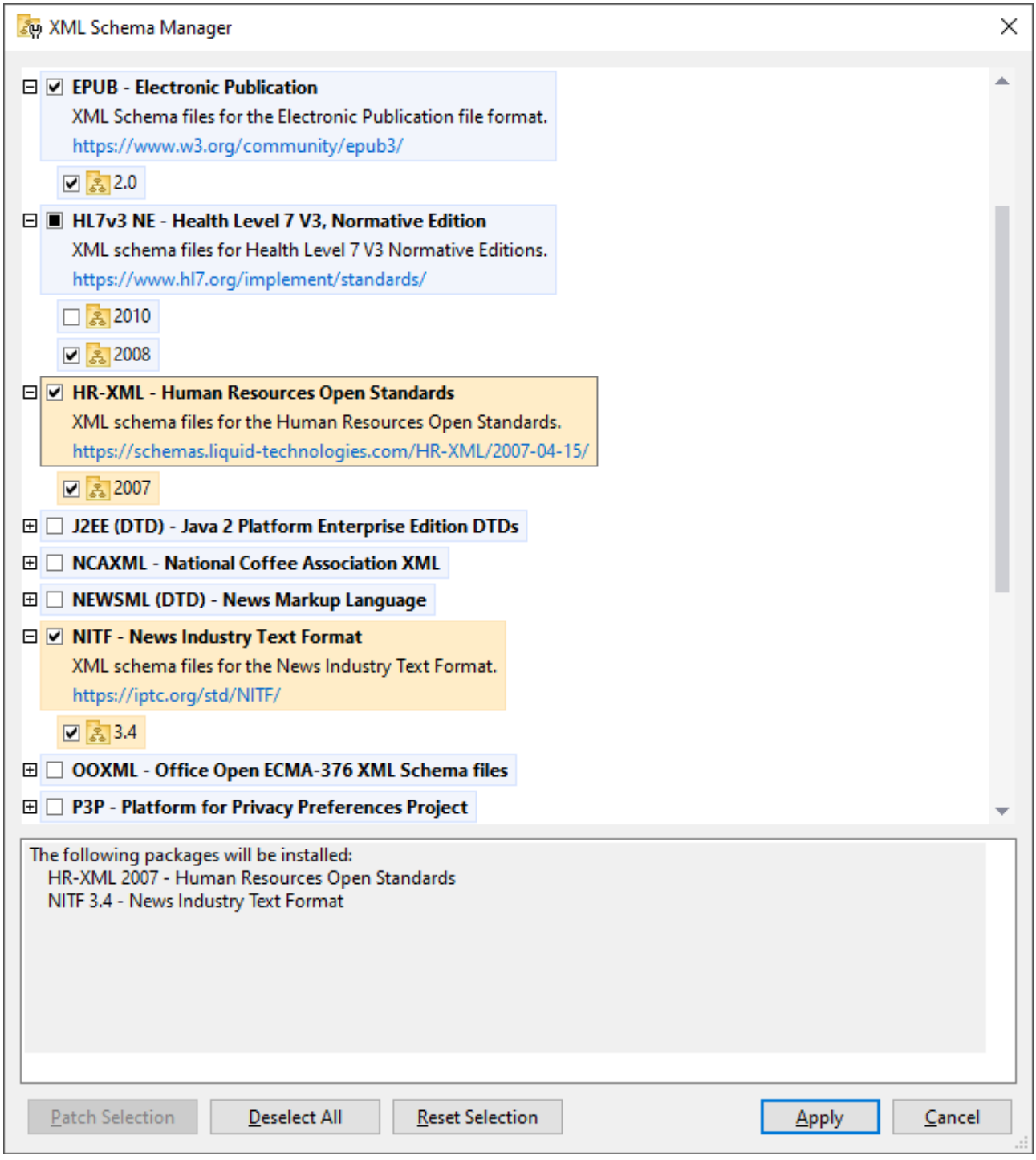
8.1 Run Schema Manager

Graphical User Interface

You can access the GUI of Schema Manager in any of the following ways:

- *During the installation of RaptorXML Server:* Towards the end of the installation procedure, select the check box *Invoke Altova XML-Schema Manager* to access the Schema Manager GUI straight away. This will enable you to install schemas during the installation process of your Altova application.
- Via the `.altova_xmlschemas` file downloaded from the Altova website: Double-click the downloaded file to run the Schema Manager GUI, which will be set up to install the schemas you selected (at the website) for installation.

After the Schema Manager GUI (*screenshot below*) has been opened, already installed schemas will be shown selected. If you want to install an additional schema, select it. If you want to uninstall an already installed schema, deselect it. After you have made your selections and/or deselections, you are ready to apply your changes. The schemas that will be installed or uninstalled will be highlighted and a message about the upcoming changes will be posted to the Messages pane at the bottom of the Schema Manager window (*see screenshot*).



Command line interface

You can run Schema Manager from a command line interface by sending commands to its executable file, `xmlschemamanager.exe`.

© 2017-2023 Altova GmbH

The **xmlschemamanager.exe** file is located in the following folder:

- *On Windows:* `C:\ProgramData\Altova\SharedBetweenVersions`
- *On Linux or macOS (server applications only):* `%INSTALLDIR%/bin`, where `%INSTALLDIR%` is the program's installation directory.

You can then use any of the commands listed in the CLI command reference section 375.

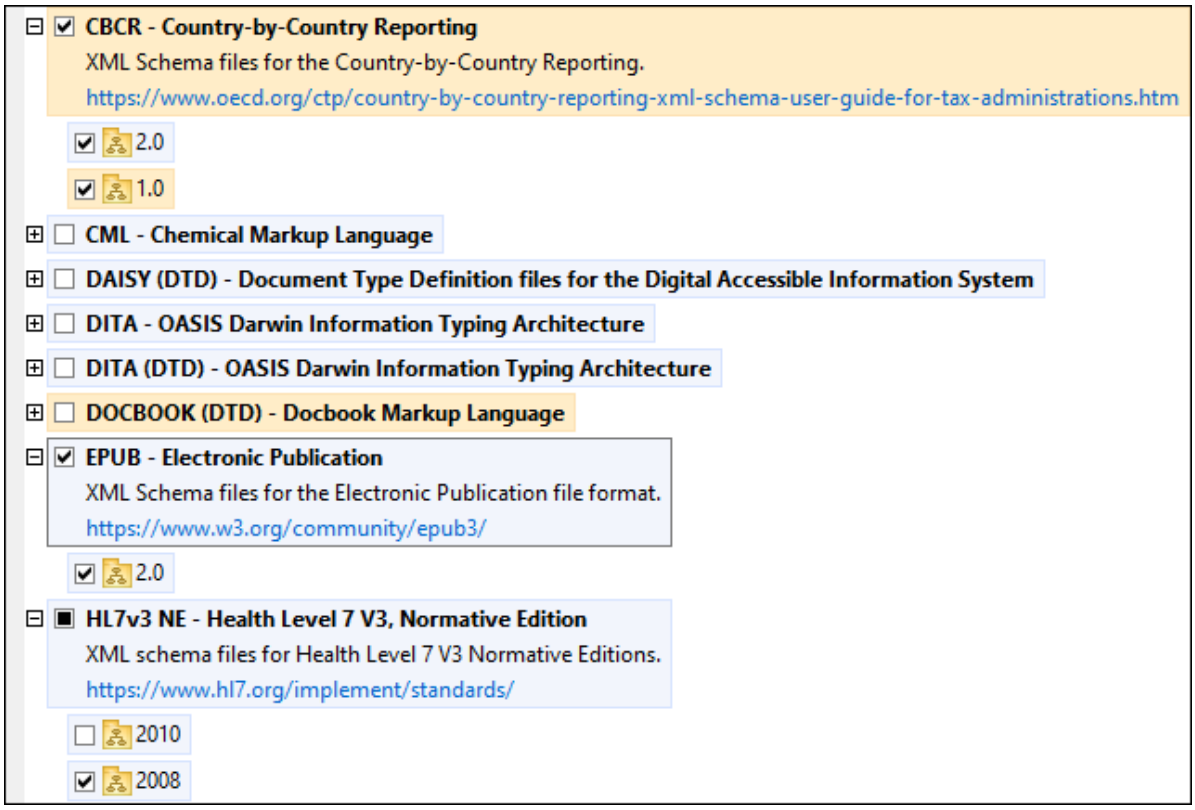
To display help for the commands, run the following:

- *On Windows:* `xmlschemamanager.exe --help`
- *On Linux or macOS (server applications only):* `sudo ./xmlschemamanager --help`

8.2 Status Categories

Schema Manager categorizes the schemas under its management as follows:

- *Installed schemas.* These are shown in the GUI with their check boxes selected (*in the screenshot below the checked and blue versions of the EPUB and HL7v3 NE schemas are installed schemas*). If all the versions of a schema are selected, then the selection mark is a tick. If at least one version is unselected, then the selection mark is a solid colored square. You can deselect an installed schema to **uninstall** it; (*in the screenshot below, the DocBook DTD is installed and has been deselected, thereby preparing it for de-installation*).
- *Uninstalled available schemas.* These are shown in the GUI with their check boxes unselected. You can select the schemas you want to **install**.



- *Upgradeable schemas* are those which have been revised by their issuers since they were installed. They are indicated in the GUI by a icon. You can **patch** an installed schema with an available revision.

Points to note

- In the screenshot above, both CBCR schemas are checked. The one with the blue background is already installed. The one with the yellow background is uninstalled and has been selected for installation. Note that the HL7v3 NE 2010 schema is not installed and has not been selected for installation.
- A yellow background means that the schema will be modified in some way when the **Apply** button is clicked. If a schema is unchecked and has a yellow background, it means that it will be uninstalled when the **Apply** button is clicked. In the screenshot above the DocBook DTD has such a status.

 © 2017-2023 Altova GmbH

- When running Schema Manager from the command line, the `list` 377 command is used with different options to list different categories of schemas:

`xmlschemamanager.exe list`	Lists all installed and available schemas; upgradeables are also indicated
`xmlschemamanager.exe list -i`	Lists installed schemas only; upgradeables are also indicated
`xmlschemamanager.exe list -u`	Lists upgradeable schemas

Note: On Linux and macOS, **`use sudo ./xmlschemamanager list`**

8.3 Patch or Install a Schema

Patch an installed schema

Occasionally, XML schemas may receive patches (upgrades or revisions) from their issuers. When Schema Manager detects that patches are available, these are indicated in the schema listings of Schema Manager and you can install the patches quickly.

In the GUI

Patches are indicated by the icon. (*Also see the previous topic about status categories*[370].) If patches are available, the **Patch Selection** button will be enabled. Click it to select and prepare all patches for installation. In the GUI, the icon of each schema that will be patched changes from to , and the Messages pane at the bottom of the dialog lists the patches that will be applied. When you are ready to install the selected patches, click **Apply**. All patches will be applied together. Note that if you deselect a schema marked for patching, you will actually be uninstalling that schema.

On the CLI

To apply a patch at the command line interface:

1. Run the `list -u`[377] command. This lists any schemas for which upgrades are available.
2. Run the `upgrade`[380] command to install all the patches.

Install an available schema

You can install schemas using either the Schema Manager GUI or by sending Schema Manager the install instructions via the command line.

Note: If the current schema references other schemas, the referenced schemas are also installed.

In the GUI

To install schemas using the Schema Manager GUI, select the schemas you want to install and click **Apply**.

You can also select the schemas you want to install at the Altova website and generate a downloadable `.altova_xmlschemas` file. When you double-click this file, it will open Schema Manager with the schemas you wanted pre-selected. All you will now have to do is click **Apply**.

On the CLI

To install schemas via the command line, run the `install`[377] command:

```
xmlschemamanager.exe install [options] Schema+
```

where `Schema` is the schema (or schemas) you want to install or a `.altova_xmlschemas` file. A schema is referenced by an identifier of format `<name>-<version>`. (The identifiers of schemas are displayed when you run the `list`[377] command.) You can enter as many schemas as you like. For details, see the description of the `install`[377] command.

Note: On Linux or macOS, use the `sudo ./xmlschemamanager` command.

© 2017-2023 Altova GmbH

Installing a required schema
When you run an XML-schema-related command in RaptorXML Server and RaptorXML Server discovers that a schema it needs for executing the command is not present or is incomplete, Schema Manager will display information about the missing schema/s. You can then directly install any missing schema via Schema Manager.

In the Schema Manager GUI, you can view all previously installed schemas at any time by running Schema Manager from **Tools | Schema Manager**.

8.4 Uninstall a Schema, Reset

Uninstall a schema

You can uninstall schemas using either the Schema Manager GUI or by sending Schema Manager the uninstall instructions via the command line.

Note: If the schema you want to uninstall references other schemas, then the referenced schemas are also uninstalled.

In the GUI

To uninstall schemas in the Schema Manager GUI, clear their check boxes and click **Apply**. The selected schemas and their referenced schemas will be uninstalled.

To uninstall all schemas, click **Deselect All** and click **Apply**.

On the CLI

To uninstall schemas via the command line, run the `uninstall` [379] command:

```
xmlschemamanager.exe uninstall [options] Schema+
```

where each **`Schema`** argument is a schema you want to uninstall or a **`.altova_xmlschemas`** file. A schema is specified by an identifier that has a format of **`<name>-<version>`**. (The identifiers of schemas are displayed when you run the `list` [377] command.) You can enter as many schemas as you like. For details, see the description of the `uninstall` [379] command.

Note: On Linux or macOS, use the **`sudo ./xmlschemamanager`** command.

Reset Schema Manager

You can reset Schema Manager. This removes all installed schemas and the cache directory.

- In the GUI, click **Reset Selection**.
- On the CLI, run the `reset` [378] command.

After running this command, make sure to run the `initialize` [376] command in order to recreate the cache directory. Alternatively, run the `reset` [378] command with the `-i` option.

Note that `reset -i` [378] restores the original installation of the product, so it is recommended to run the `update` [380] command after performing a reset. Alternatively, run the `reset` [378] command with the `-i` and `-u` options.

© 2017-2023 Altova GmbH

8.5 Command Line Interface (CLI)

To call Schema Manager at the command line, you need to know the path of the executable. By default, the Schema Manager executable is installed here:

Windows	`C:\ProgramData\Altova\SharedBetweenVersions\XMLSchemaManager.exe`
Linux	`/opt/Altova/RaptorXMLServer2024/bin/xmlschemamanager`
macOS	`/usr/local/Altova/RaptorXMLServer2024/bin/xmlschemamanager`

Note: On Linux and macOS systems, once you have changed the directory to that containing the executable, you can call the executable with **`sudo ./xmlschemamanager`**. The prefix **`./`** indicates that the executable is in the current directory. The prefix **`sudo`** indicates that the command must be run with root privileges.

Command line syntax

The general syntax for using the command line is as follows:

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

In the listing above, the vertical bar **`|`** separates a set of mutually exclusive items. The square brackets **`[]`** indicate optional items. Essentially, you can type the executable path followed by either `--h`, `--help`, or `--version` options, or by a command. Each command may have options and arguments. The list of commands is described in the following sections.

8.5.1 help

This command provides contextual help about commands pertaining to Schema Manager executable.

Syntax

```
<exec> help [command]
```

Where `[command]` is an optional argument which specifies any valid command name.

Note the following:

- You can invoke help for a command by typing the command followed by **`-h`** or **`--help`**, for example: **`<exec> list -h`**
- If you type **`-h`** or **`--help`** directly after the executable and before a command, you will get general help (not help for the command), for example: **`<exec> -h list`**

Example

The following command displays help about the `list` command:

```
xmlschemamanager help list
```

8.5.2 info

This command displays detailed information for each of the schemas supplied as a `Schema` argument. This information for each submitted schema includes the title, version, description, publisher, and any referenced schemas, as well as whether the schema has been installed or not.

Syntax

```
<exec> info [options] Schema+
```

- The **`Schema`** argument is the name of a schema or a part of a schema's name. (To display a schema's package ID and detailed information about its installation status, you should use the `list` 377 command.)
- Use **`<exec> info -h`** to display help for the command.

Example

The following command displays information about the latest **`DocBook-DTD`** and **`NITF`** schemas:

```
xmlschemamanager info doc nitf
```

8.5.3 initialize

This command initializes the Schema Manager environment. It creates a cache directory where information about all schemas is stored. Initialization is performed automatically the first time a schema-cognizant Altova application is installed. You would not need to run this command under normal circumstances, but you would typically need to run it after executing the **`reset`** command.

Syntax

```
<exec> initialize | init [options]
```

Options

The **`initialize`** command takes the following options:

`--silent, --s`	Display only error messages. The default is **`false`**.
`--verbose, --v`	Display detailed information during execution. The default is **`false`**.
`--help, --h`	Display help for the command.

Example

The following command initializes Schema Manager:

© 2017-2023 Altova GmbH

```
xmlschemamanager initialize
```

8.5.4 install

This command installs one or more schemas.

Syntax

```
<exec> install [options] Schema+
```

To install multiple schemas, add the **Schema** argument multiple times.

The **Schema** argument is one of the following:

- A schema identifier (having a format of **<name>-<version>**, for example: **cbcr-2.0**). To find out the schema identifiers of the schemas you want, run the list 377 command. You can also use an abbreviated identifier if it is unique, for example **docbook**. If you use an abbreviated identifier, then the latest version of that schema will be installed.
- The path to a **.altova_xmlschemas** file downloaded from the Altova website. For information about these files, see *Introduction to SchemaManager: How It Works* 363.

Options
The **install** command takes the following options:

`--silent, --s`	Display only error messages. The default is **false**.
`--verbose, --v`	Display detailed information during execution. The default is **false**.
`--help, --h`	Display help for the command.

Example

The following command installs the CBCR 2.0 (Country-By-Country Reporting) schema and the latest DocBook DTD:

```
xmlschemamanager install cbcr-2.0 docbook
```

8.5.5 list

This command lists schemas under the management of Schema Manager. The list displays one of the following

- All available schemas
- Schemas containing in their name the string submitted as a **Schema** argument
- Only installed schemas
- Only schemas that can be upgraded

Syntax

```
<exec> list | ls [options] Schema?
```

If no **Schema** argument is submitted, then all available schemas are listed. Otherwise, schemas are listed as specified by the submitted options (*see example below*). Note that you can submit the **Schema** argument multiple times.

Options
The **list** command takes the following options:

`--installed, --i`	List only installed schemas. The default is **false**.
`--upgradeable, --u`	List only schemas where upgrades (patches) are available. The default is **false**.
`--help, --h`	Display help for the command.

Examples

- To list all available schemas, run: **xmlschemamanager list**
- To list installed schemas only, run: **xmlschemamanager list -i**
- To list schemas that contain either "doc" or "nitf" in their name, run: **xmlschemamanager list doc nitf**

8.5.6 reset

This command removes all installed schemas and the cache directory. You will be completely resetting your schema environment. After running this command, be sure to run the initialize 376 command to recreate the cache directory. Alternatively, run the **reset** command with the **-i** option. Since **reset -i** restores the original installation of the product, we recommend that you run the update 380 command after performing a reset and initialization. Alternatively, run the **reset** command with both the **-i** and **-u** options.

Syntax

```
<exec> reset [options]
```

Options
The **reset** command takes the following options:

`--init, --i`	Initialize Schema Manager after reset. The default is **false**.
`--update, --u`	Updates the list of available schemas in the cache. The default is **false**.
`--silent, --s`	Display only error messages. The default is **false**.
`--verbose, --v`	Display detailed information during execution. The default is **false**.

 © 2017-2023 Altova GmbH

`--help, --h`	Display help for the command.

Examples

- To reset Schema Manager, run: **`xmlschemamanager reset`**
- To reset Schema Manager and initialize it, run: **`xmlschemamanager reset -i`**
- To reset Schema Manager, initialize it,and update its schema list, run: **`xmlschemamanager reset -i -u`**

8.5.7 uninstall

This command uninstalls one or more schemas. By default, any schemas referenced by the current one are uninstalled as well. To uninstall just the current schema and keep the referenced schemas, set the option `--k`.

Syntax

```
<exec> uninstall [options] Schema+
```

To uninstall multiple schemas, add the **`Schema`** argument multiple times.

The **`Schema`** argument is one of the following:

- A schema identifier (having a format of **`<name>-<version>`**, for example: **`cbcr-2.0`**). To find out the schema identifiers of the schemas that are installed, run the `list -i` 377 command. You can also use an abbreviated schema name if it is unique, for example **`docbook`**. If you use an abbreviated name, then all schemas that contain the abbreviation in its name will be uninstalled.
- The path to a **`.altova_xmlschemas`** file downloaded from the Altova website. For information about these files, see *Introduction to SchemaManager: How It Works* 363.

Options
The **`uninstall`** command takes the following options:

`--keep-references, --k`	Set this option to keep referenced schemas. The default is **`false`**.
`--silent, --s`	Display only error messages. The default is **`false`**.
`--verbose, --v`	Display detailed information during execution. The default is **`false`**.
`--help, --h`	Display help for the command.

Example

The following command uninstalls the CBCR 2.0 and EPUB 2.0 schemas and their dependencies:

```
xmlschemamanager uninstall cbcr-2.0 epub-2.0
```

The following command uninstalls the **`eba-2.10`** schema but not the schemas it references:

```
xmlschemamanager uninstall --k cbcr-2.0
```

8.5.8 update

This command queries the list of schemas available from the online storage and updates the local cache directory. You should not need to run this command unless you have performed a `reset`[378] and `initialize`[376].

Syntax

```
<exec> update [options]
```

Options
The **update** command takes the following options:

`--silent, --s`	Display only error messages. The default is **false**.
`--verbose, --v`	Display detailed information during execution. The default is **false**.
`--help, --h`	Display help for the command.

Example

The following command updates the local cache with the list of latest schemas:

```
xmlschemamanager update
```

8.5.9 upgrade

This command upgrades all installed schemas that can be upgraded to the latest available *patched* version. You can identify upgradeable schemas by running the `list -u`[377] command.

Note: The **upgrade** command removes a deprecated schema if no newer version is available.

Syntax

```
<exec> upgrade [options]
```

Options
The **upgrade** command takes the following options:

`--silent, --s`	Display only error messages. The default is **false**.
`--verbose, --v`	Display detailed information during execution. The default is **false**.
`--help, --h`	Display help for the command.

 © 2017-2023 Altova GmbH

9 Additional Information

This section contains the following additional information:

- Exit Codes 382
- Schema Location Hints 383

9.1 Exit Codes

The following exit codes are available:

0	Validation successful
1	Validation failed with error / Process interrupted by Ctrl+C/Break/terminal closed / License expired during execution
11	RaptorXML could not start; the reason is given in the log file
22	Could not load root catalog / Could not load list file
64	Invalid command/options
77	Failed to acquire license during startup
128+**n**	RaptorXML terminated because of signal number **n**. All exit codes above **128** indicate termination as a result of a received external signal or an internally triggered signal. For example, if the exit code is **134**, then the signal number is **134-128=6** (the number of **SIGABRT**).

© 2017-2023 Altova GmbH

9.2 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 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` 225 option (for XBRL and XSD/XML) specifies how the schema location is to be selected.

10 Engine Information

This section contains information about the XSLT and XQuery engines contained in RaptorXML Server. This information mostly concerns engine behavior in situations where the specifications leave the decision regarding behavior up to the implementation. This section also contains information about Altova extension functions for XPath/XQuery.

© 2017-2023 Altova GmbH

10.1 XSLT and XQuery Engine Information

The XSLT and XQuery engines of RaptorXML 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 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:

- XSLT 1.0 385
- XSLT 2.0 385
- XSLT 3.0 387
- XQuery 1.0 388
- XQuery 3.1 391

10.1.1 XSLT 1.0

The XSLT 1.0 Engine of RaptorXML 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 U+00A0 (the hexadecimal character reference for a non-breaking space) is inserted in the HTML code either as a character reference (` ` or ` `) or as an entity reference, ` `.

10.1.2 XSLT 2.0

This section:

- Engine conformance 386
- Backward compatibility 386
- Namespaces 386
- Schema awareness 386

- Implementation-specific behavior 387

Conformance

The XSLT 2.0 engine of RaptorXML 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. Typically, the backwards compatibility of the XSLT 2.0 engine comes into play when using the XSLT 2.0 engine (CLI parameter `--xslt=2` 229) to process an XSLT 1.0 stylesheet or instruction. 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.

© 2017-2023 Altova GmbH

Implementation-specific behavior

Given below is a description of how the XSLT 2.0 engine handles implementation-specific aspects 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, XPath, and extension functions).

unparsed-text
The `href` argument 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`. Example: `xs:base64Binary(unparsed-text('chart.png', 'x-binarytobase64'))`.

unparsed-text-available
The `href` argument 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`.

10.1.3 XSLT 3.0

The XSLT 3.0 Engine of RaptorXML Server conforms to the World Wide Web Consortium's (W3C's) XSLT 3.0 Recommendation of 8 June 2017 and XPath 3.1 Recommendation of 21 March 2017.

The XSLT 3.0 engine has the same implementation-specific characteristics as the XSLT 2.0 engine 385. Additionally, it includes support for a number of new XSLT 3.0 features: XPath/XQuery 3.1 functions and operators, and the XPath 3.1 specification.

Note: The optional streaming feature is not supported currently. The entire document will be loaded into memory regardless of the value of the `streamable` attribute. If enough memory is available, then: (i) the entire document will be processed—without streaming, (ii) guaranteed-streamable constructs will be processed correctly, as if the execution used streaming, and (iii) streaming errors will not be detected. In 64-bit apps, non-streaming execution should not be a problem. If memory does turn out to be an issue, a solution would be to add more memory to the system.

Namespaces

Your XSLT 3.0 stylesheet should declare the following namespaces in order for you to be able to use all the type constructors and functions available in XSLT 3.0. 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/XQuery 3.1 functions	`fn:`	`http://www.w3.org/2005/xpath-functions`
Math functions	`math:`	`http://www.w3.org/2005/xpath-functions/math`
Map functions	`map:`	`http://www.w3.org/2005/xpath-functions/map`
Array functions	`array:`	`http://www.w3.org/2005/xpath-functions/array`
XQuery, XSLT, and XPath Error Codes	`err:`	`http://www.w3.org/2005/xpath-functions/xqt-errors`
Serialization functions	`output`	`http://www.w3.org/2010/xslt-xquery-serialization`

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

```
<xsl:stylesheet version="3.0"
   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 3.0 engine uses the XPath and XQuery Functions and Operators 3.1 namespace (listed in the table above) as its **default functions namespace**. So you can use the functions of this namespace in your stylesheet without any prefix. If you declare the 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/XQuery 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')`.

10.1.4 XQuery 1.0

This section:

- Engine conformance 389
- Schema awareness 389
- Encoding 389
- Namespaces 386
- XML source and validation 390
- Static and dynamic type checking 390
- Library modules 390
- External functions 390
- Collations 390

 © 2017-2023 Altova GmbH

- Precision of numeric data 391
- XQuery instructions support 391
- Implementation-specific behavior 391

Conformance

The XQuery 1.0 Engine of RaptorXML 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 (see `fn:`) 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[393]. To use a specific collation, supply

 © 2017-2023 Altova GmbH

its URI as given in the list of supported collations 393. 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.

Implementation-specific behavior

Given below is a description of how the XQuery and XQuery Update 1.0 engines handle implementation-specific aspects of certain functions.

`unparsed-text`
The `href` argument 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`. Example: `xs:base64Binary(unparsed-text('chart.png', 'x-binarytobase64'))`.

`unparsed-text-available`
The `href` argument 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`.

10.1.5 XQuery 3.1

The XQuery 3.1 Engine of RaptorXML Server conforms to the World Wide Web Consortium's (W3C's) XQuery 3.1 Recommendation of 21 March 2017 and includes support for XPath and XQuery Functions 3.1. The XQuery 3.1 specification is a superset of the 3.0 specification. The XQuery 3.1 engine therefore supports XQuery 3.0 features.

Namespaces

Your XQuery 3.1 document should declare the following namespaces in order for you to be able to use all the type constructors and functions available in XQuery 3.1. 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/XQuery 3.1 functions	fn:	http://www.w3.org/2005/xpath-functions
Math functions	math:	http://www.w3.org/2005/xpath-functions/math
Map functions	map:	http://www.w3.org/2005/xpath-functions/map
Array functions	array:	http://www.w3.org/2005/xpath-functions/array
XQuery, XSLT, and XPath Error Codes	err:	http://www.w3.org/2005/xpath-functions/xqt-errors
Serialization functions	output	http://www.w3.org/2010/xslt-xquery-serialization

The following points should be noted:

- The XQuery 3.1 Engine recognizes the prefixes listed above as being bound to the corresponding namespaces.
- Since the built-in functions namespace listed above (see `fn:`) 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")`.)

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.

Implementation-specific behavior

Implementation-specific characteristics are the same as for XQuery 1.0 388.

Additionally, the Altova-specific encoding `x-base64tobinary` can be used to create a binary result document, such as an image.

 © 2017-2023 Altova GmbH

10.2 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.

This section mainly describes XPath/XQuery extension functions that have been created by Altova to provide additional operations. These functions 394 can be computed by Altova's XSLT and XQuery engines according to the rules described in this section. For information about the regular XPath/XQuery functions, see Altova's XPath/XQuery Function Reference.

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 Altova XSLT and XQuery engines read non-prefixed functions as belonging to the namespace `http://www.w3.org/2005/xpath-functions`, which is the default functions namespace specified in the XPath/XQuery functions specifications. 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 timezones 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. The engine uses the Unicode Collation Algorithm. Other supported collations are the ICU collations listed below; to use one of these, supply its URI as given in the 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.

10.2.1 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

 © 2017-2023 Altova GmbH

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 XP3.1
XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3
XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3.1

Usage of Altova extension functions

In order to use Altova extension functions, you must declare the Altova extension functions namespace (*first highlight in code listing below*) and then use the extension functions so that they are resolved as belonging to this namespace (*see second highlight*). The example below uses the Altova extension function named **age**.

```
<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/xslt-extensions">
    <xsl:output method="text" encoding="ISO-8859-1"/>
    <xsl:template match="Persons">
        <xsl:for-each select="Person">
            <xsl:value-of select="concat(Name, ': ')"/>
            <xsl:value-of select="altova:age(xs:date(BirthDate))"/>
            <xsl:value-of select="' years&#x0A;'"/>
        </xsl:for-each>
    </xsl:template>
</xsl:stylesheet>
```

XSLT functions 396

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 can be used both in XPath expressions in XSLT contexts as well as in XQuery expressions:

- Date/Time 399
- Geolocation 416
- Image-related 427
- Numeric 432
- Sequence 453
- String 461

- Miscellaneous 468

Chart functions (Enterprise and Server Editions only)

Altova extension functions for charts 470 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 482 enable barcodes to be generated and placed in output generated via XSLT stylesheets.

10.2.1.1 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 XP3.1
XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3
XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3.1

General 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

 © 2017-2023 Altova GmbH

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 396]

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 396]

 © 2017-2023 Altova GmbH

10.2.1.2 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 XP3.1
XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3
XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3.1

▼ Grouped by functionality

- Add a duration to xs:dateTime and return xs:dateTime 400
- Add a duration to xs:date and return xs:date 402
- Add a duration to xs:time and return xs:time 403
- Format and retrieve durations 403
- Remove timezone from functions that generate current date/time 404
- Return days, hours, minutes, and seconds from durations 406
- Return weekday as integer from date 407
- Return week number as integer from date 407
- Build date, time, or duration type from lexical components of each type 410
- Construct date, dateTime, or time type from string input 411
- Age-related functions 413
- Epoch time (Unix time) functions 414

▼ Listed alphabetically

altova:add-days-to-date 402
altova:add-days-to-dateTime 400
altova:add-hours-to-dateTime 400
altova:add-hours-to-time 403
altova:add-minutes-to-dateTime 400
altova:add-minutes-to-time 403
altova:add-months-to-date 402
altova:add-months-to-dateTime 400
altova:add-seconds-to-dateTime 400
altova:add-seconds-to-time 403
altova:add-years-to-date 402
altova:add-years-to-dateTime 400
altova:age 413

altova:age-details 413
altova:build-date 410
altova:build-duration 410
altova:build-time 410
altova:current-dateTime-no-TZ 404
altova:current-date-no-TZ 404
altova:current-time-no-TZ 404
altova:date-no-TZ 404
altova:dateTime-from-epoch 414
altova:dateTime-from-epoch-no-TZ 414
altova:dateTime-no-TZ 404
altova:days-in-month 406
altova:epoch-from-dateTime 414
altova:hours-from-dateTimeDuration-accumulated 406
altova:minutes-from-dateTimeDuration-accumulated 406
altova:seconds-from-dateTimeDuration-accumulated 406
altova:format-duration 403
altova:parse-date 411
altova:parse-dateTime 411
altova:parse-duration 403
altova:parse-time 411
altova:time-no-TZ 404
altova:weekday-from-date 407
altova:weekday-from-dateTime 407
altova:weeknumber-from-date 408
altova:weeknumber-from-dateTime 408

[Top 399]

Add a duration to xs:dateTime XP3.1 XQ3.1

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** XP3.1 XQ3.1

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** XP3.1 XQ3.1

 © 2017-2023 Altova GmbH

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** **XP3.1** **XQ3.1**

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** **XP3.1** **XQ3.1**

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** **XP3.1** **XQ3.1**

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** XP3.1 XQ3.1

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 399]

Add a duration to xs:date XP3.1 XQ3.1

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** XP3.1 XQ3.1

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** XP3.1 XQ3.1

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** XP3.1 XQ3.1

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

 © 2017-2023 Altova GmbH

[Top 399]

Format and retrieve durations XP3.1 XQ3.1

These functions parse an input **xs:duration** or **xs:string** and return, respectively, an **xs:string** or **xs:duration**.

▼ format-duration [altova:]

altova:format-duration(Duration *as xs:duration*, **Picture** *as xs:string*) **as xs:string** XP3.1 XQ3.1

Formats a duration, which is submitted as the first argument, according to a picture string submitted as the second argument. The output is a text string formatted according to the picture string.

Examples

- **altova:format-duration**(xs:duration("P2DT2H53M11.7S"), "Days:[D01] Hours:[H01] Minutes:[m01] Seconds:[s01] Fractions:[f0]") returns "Days:02 Hours:02 Minutes:53 Seconds:11 Fractions:7"
- **altova:format-duration**(xs:duration("P3M2DT2H53M11.7S"), "Months:[M01] Days:[D01] Hours:[H01] Minutes:[m01]") returns "Months:03 Days:02 Hours:02 Minutes:53"

▼ parse-duration [altova:]

altova:parse-duration(InputString *as xs:string*, **Picture** *as xs:string*) **as xs:duration** XP3.1 XQ3.1

Takes a patterned string as the first argument, and a picture string as the second argument. The input string is parsed on the basis of the picture string, and an xs:duration is returned.

Examples

- **altova:parse-duration**("Days:02 Hours:02 Minutes:53 Seconds:11 Fractions:7", "Days:[D01] Hours:[H01] Minutes:[m01] Seconds:[s01] Fractions:[f0]") returns "P2DT2H53M11.7S"
- **altova:parse-duration**("Months:03 Days:02 Hours:02 Minutes:53 Seconds:11 Fractions:7", "Months:[M01] Days:[D01] Hours:[H01] Minutes:[m01]") returns "P3M2DT2H53M"

[Top 399]

Add a duration to xs:time XP3.1 XQ3.1

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** XP3.1 XQ3.1

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`** XP3.1 XQ3.1

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`** XP3.1 XQ3.1

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 399]

Remove the timezone part from date/time datatypes XP3.1 XQ3.1

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-date-no-TZ [altova:]

`altova:current-date-no-TZ() as xs:date` XP3.1 XQ3.1

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`

 © 2017-2023 Altova GmbH

▼ current-dateTime-no-TZ [altova:]

altova:current-dateTime-no-TZ() as xs:dateTime XP3.1 XQ3.1
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-time-no-TZ [altova:]

altova:current-time-no-TZ() as xs:time XP3.1 XQ3.1
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`

▼ date-no-TZ [altova:]

altova:date-no-TZ(InputDate *as xs:date*) **as xs:date** XP3.1 XQ3.1
This function takes an `xs:date` argument, removes the timezone part from it, and returns an `xs:date` value. Note that the date is not modified.

Examples

- **altova:date-no-TZ**(xs:date("2014-01-15+01:00")) returns `2014-01-15`

▼ dateTime-no-TZ [altova:]

altova:dateTime-no-TZ(InputDateTime *as xs:dateTime*) **as xs:dateTime** XP3.1 XQ3.1
This function takes an `xs:dateTime` argument, removes the timezone part from it, and returns an `xs:dateTime` value. Note that neither the date nor the time is modified.

Examples

- **altova:dateTime-no-TZ**(xs:date("2014-01-15T14:00:00+01:00")) returns `2014-01-15T14:00:00`

▼ time-no-TZ [altova:]

altova:time-no-TZ(InputTime *as xs:time*) **as xs:time** XP3.1 XQ3.1
This function takes an `xs:time` argument, removes the timezone part from it, and returns an `xs:time` value. Note that the time is not modified.

Examples

- **altova:time-no-TZ**(xs:time("14:00:00+01:00")) returns 14:00:00

[Top 399]

Return the number of days, hours, minutes, seconds from durations XP3.1 XQ3.1

These functions return the number of days in a month, and the number of hours, minutes, and seconds, respectively, from durations.

days-in-month [altova:]

altova:days-in-month(Year *as xs:integer*, **Month** *as xs:integer*) **as xs:integer** XP3.1 XQ3.1
Returns the number of days in the specified month. The month is specified by means of the Year and Month arguments.

Examples

- **altova:days-in-month**(2018, 10) returns 31
- **altova:days-in-month**(2018, 2) returns 28
- **altova:days-in-month**(2020, 2) returns 29

hours-from-dayTimeDuration-accumulated

altova:hours-from-dayTimeDuration-accumulated(DayAndTime *as xs:duration*) **as xs:integer** XP3.1 XQ3.1
Returns the total number of hours in the duration submitted by the DayAndTime argument (which is of type xs:duration). The hours in the Day and Time components are added together to give a result that is an integer. A new hour is counted only for a full 60 minutes. Negative durations result in a negative hour value.

Examples

- **altova:hours-from-dayTimeDuration-accumulated**(xs:duration("P5D")) returns 120, which is the total number of hours in 5 days.
- **altova:hours-from-dayTimeDuration-accumulated**(xs:duration("P5DT2H")) returns 122, which is the total number of hours in 5 days plus 2 hours.
- **altova:hours-from-dayTimeDuration-accumulated**(xs:duration("P5DT2H60M")) returns 123, which is the total number of hours in 5 days plus 2 hours and 60 mins.
- **altova:hours-from-dayTimeDuration-accumulated**(xs:duration("P5DT2H119M")) returns 123, which is the total number of hours in 5 days plus 2 hours and 119 mins.
- **altova:hours-from-dayTimeDuration-accumulated**(xs:duration("P5DT2H120M")) returns 124, which is the total number of hours in 5 days plus 2 hours and 120 mins.
- **altova:hours-from-dayTimeDuration-accumulated**(xs:duration("-P5DT2H")) returns -122

minutes-from-dayTimeDuration-accumulated

altova:minutes-from-dayTimeDuration-accumulated(DayAndTime *as xs:duration*) **as xs:integer** XP3.1 XQ3.1
Returns the total number of minutes in the duration submitted by the DayAndTime argument (which is of type xs:duration). The minutes in the Day and Time components are added together to give a result that is an integer. Negative durations result in a negative minute value.

Examples

© 2017-2023 Altova GmbH

- **altova:minutes-from-dayTimeDuration-accumulated**(xs:duration("PT60M")) returns 60
- **altova:minutes-from-dayTimeDuration-accumulated**(xs:duration("PT1H")) returns 60, which is the total number of minutes in 1 hour.
- **altova:minutes-from-dayTimeDuration-accumulated**(xs:duration("PT1H40M")) returns 100
- **altova:minutes-from-dayTimeDuration-accumulated**(xs:duration("P1D")) returns 1440, which is the total number of minutes in 1 day.
- **altova:minutes-from-dayTimeDuration-accumulated**(xs:duration("-P1DT60M")) returns -1500

▼ seconds-from-dayTimeDuration-accumulated

altova:seconds-from-dayTimeDuration-accumulated(DayAndTime *as xs:duration*) **as xs:integer** XP3.1 XQ3.1

Returns the total number of seconds in the duration submitted by the DayAndTime argument (which is of type xs:duration). The seconds in the Day and Time components are added together to give a result that is an integer. Negative durations result in a negative seconds value.

Examples

- **altova:seconds-from-dayTimeDuration-accumulated**(xs:duration("PT1M")) returns 60, which is the total number of seconds in 1 minute.
- **altova:seconds-from-dayTimeDuration-accumulated**(xs:duration("PT1H")) returns 3600, which is the total number of seconds in 1 hour.
- **altova:seconds-from-dayTimeDuration-accumulated**(xs:duration("PT1H2M")) returns 3720
- **altova:seconds-from-dayTimeDuration-accumulated**(xs:duration("P1D")) returns 86400, which is the total number of seconds in 1 day.
- **altova:seconds-from-dayTimeDuration-accumulated**(xs:duration("-P1DT1M")) returns -86460

Return the weekday from xs:dateTime or xs:date XP3.1 XQ3.1

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** XP3.1 XQ3.1

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** XP3.1 XQ3.1

Takes a date-with-time as its first argument and returns the day of the week of this date as an integer. 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** XP3.1 XQ3.1

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** XP3.1 XQ3.1

Takes a date as its first argument and returns the day of the week of this date as an integer. 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 399]

Return the week number from xs:dateTime or xs:date XP2 XQ1 XP3.1 XQ3.1

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 XP3.1 XQ3.1

© 2017-2023 Altova GmbH

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 XP3.1 XQ3.1

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 399]

Build date, time, and duration datatypes from their lexical components XP3.1 XQ3.1

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** XP3.1 XQ3.1

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** XP3.1 XQ3.1

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** XP3.1 XQ3.1

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** XP3.1 XQ3.1

Takes two arguments to build a value of type `xs:yearMonthDuration`. The first argument 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

© 2017-2023 Altova GmbH

- **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** **XP3.1** **XQ3.1**
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 399]

Construct date, dateTime, and time datatypes from string input **XP2** **XQ1** **XP3.1** **XQ3.1**

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** **XP3.1** **XQ3.1**
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("09-12-2014"), "[D]-[M]-[Y]") returns 2014-12-09
- **altova:parse-date**(xs:string("09-12-2014"), "[M]-[D]-[Y]") returns 2014-09-12
- **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 XP3.1 XQ3.1

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("09-12-2014 13:56:24"), "[M]-[D]-[Y] [H]:[m]:[s]") returns 2014-09-12T13:56:24
- **altova:parse-dateTime**("time=13:56:24; date=09-12-2014", "time=[H]:[m]:[s]; date=[D]-[M]-[Y]") returns 2014-12-09T13:56:24

parse-time [altova:]

altova:parse-time(Time *as xs:string*, **TimePattern** *as xs:string*) **as xs:time** XP2 XQ1 XP3.1 XQ3.1

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 399]

 © 2017-2023 Altova GmbH

Age-related functions XP3.1 XQ3.1

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** XP3.1 XQ3.1
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** XP3.1 XQ3.1
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**)* XP3.1 XQ3.1
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**)* **XP3.1** **XQ3.1**
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 399]

Epoch time (Unix time) functions XP3.1 XQ3.1

Epoch time is a time system used on Unix systems. It defines any given point in time as being the number of seconds that have elapsed since 00:00:00 UTC on 1 January 1970. Altova's Epoch time extension functions convert **xs:dateTime** values to Epoch time values and vice versa.

dateTime-from-epoch [altova:]

altova:dateTime-from-epoch(Epoch *as xs:decimal* **as xs:dateTime** **XP3.1** **XQ3.1**
Epoch time is a time system used on Unix systems. It defines any given point in time as being the number of seconds that have elapsed since 00:00:00 UTC on 1 January 1970. The **dateTime-from-epoch** function returns the **xs:dateTime** equivalent of an Epoch time, adjusts it for the local timezone, and includes the timezone information in the result.

The function takes an xs:decimal argument and returns an xs:dateTime value that includes a **TZ** (timezone) part. The result is obtained by calculating the UTC **dateTime** equivalet of the Epoch time, and adding to it the local timezone (taken from the system clock). For example, if the function is executed on a machine that has been set to be in a timezone of +01:00 (relative to UTC), then after the UTC **dateTime** equivalent has been calculated, one hour will be added to the result. The timezone information, which is an optional lexical part of the **xs:dateTime** result, is also reported in the **dateTime** result. Compare this result with that of **dateTime-from-epoch-no-TZ**, and also see the function **epoch-from-dateTime**.

Examples

The examples below assume a local timezone of UTC +01:00. Consequently, the UTC **dateTime** equivalent of the submitted Epoch time will be incremented by one hour. The timezone is reported in the result.

- **altova:dateTime-from-epoch**(34) returns 1970-01-01T01:00:34+01:00
- **altova:dateTime-from-epoch**(62) returns 1970-01-01T01:01:02+01:00

dateTime-from-epoch-no-TZ [altova:]

altova:dateTime-from-epoch-no-TZ(Epoch *as xs:decimal* **as xs:dateTime** **XP3.1** **XQ3.1**

© 2017-2023 Altova GmbH

Epoch time is a time system used on Unix systems. It defines any given point in time as being the number of seconds that have elapsed since 00:00:00 UTC on 1 January 1970. The **dateTime-from-epoch-no-TZ** function returns the **xs:dateTime** equivalent of an Epoch time, adjusts it for the local timezone, but does not include the timezone information in the result.

The function takes an `xs:decimal` argument and returns an `xs:dateTime` value that does not includes a **TZ** (timezone) part. The result is obtained by calculating the UTC **dateTime** equivalet of the Epoch time, and adding to it the local timezone (taken from the system clock). For example, if the function is executed on a machine that has been set to be in a timezone of +01:00 (relative to UTC), then after the UTC **dateTime** equivalent has been calculated, one hour will be added to the result. The timezone information, which is an optional lexical part of the **xs:dateTime** result, is not reported in the **dateTime** result. Compare this result with that of **dateTime-from-epoch**, and also see the function **epoch-from-dateTime**.

Examples

The examples below assume a local timezone of UTC +01:00. Consequently, the UTC **dateTime** equivalent of the submitted Epoch time will be incremented by one hour. The timezone is not reported in the result.

- **altova:dateTime-from-epoch**(34) returns 1970-01-01T01:00:34
- **altova:dateTime-from-epoch**(62) returns 1970-01-01T01:01:02

epoch-from-dateTime [altova:]

altova:epoch-from-dateTime(dateTimeValue *as xs:dateTime*) **as xs:decimal** XP3.1 XQ3.1

Epoch time is a time system used on Unix systems. It defines any given point in time as being the number of seconds that have elapsed since 00:00:00 UTC on 1 January 1970. The **epoch-from-dateTime** function returns the Epoch time equivalent of the **xs:dateTime** that is submitted as the argument of the function. Note that you might have to explicitly construct the **xs:dateTime** value. The submitted **xs:dateTime** value may or may not contain the optional **TZ** (timezone) part.

Whether the timezone part is submitted as part of the argument or not, the local timezone offset (taken from the system clock) is subtracted from the submitted **dateTimeValue** argument. This produces the equivalent UTC time, from which the equivalent Epoch time is calculated. For example, if the function is executed on a machine that has been set to be in a timezone of +01:00 (relative to UTC), then one hour is subtracted from the submitted **dateTimeValue** before the Epoch value is calculated. Also see the function **dateTime-from-epoch**.

Examples

The examples below assume a local timezone of UTC +01:00. Consequently, one hour will be subtracted from the submitted **dateTime** before the Epoch time is calculated.

- **altova:epoch-from-dateTime**(xs:dateTime("1970-01-01T01:00:34+01:00")) returns 34
- **altova:epoch-from-dateTime**(xs:dateTime("1970-01-01T01:00:34")) returns 34
- **altova:epoch-from-dateTime**(xs:dateTime("2021-04-01T11:22:33")) returns 1617272553

[Top 399]

10.2.1.3 XPath/XQuery Functions: Geolocation

The following geolocation XPath/XQuery extension functions are supported in the current version of RaptorXML Server 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** **XP3.1**
XSLT functions (used in XPath expressions in XSLT):	**XSLT1** **XSLT2** **XSLT3**
XQuery functions (used in XQuery expressions in XQuery):	**XQ1** **XQ3.1**

format-geolocation [altova:]

altova:format-geolocation(Latitude *as xs:decimal*, **Longitude** *as xs:decimal*, **GeolocationOutputStringFormat** *as xs:integer*) **as xs:string** **XP3.1** **XQ3.1**

Takes the latitude and longitude as the first two arguments, and outputs the geolocation as a string. The third argument, **GeolocationOutputStringFormat**, is the format of the geolocation output string; it uses integer values from `1` to `4` to identify the output string format (*see 'Geolocation output string formats' below*). Latitude values range from `+90` to `-90` (`N` to `S`). Longitude values range from `+180` to `-180` (`E` to `W`).

Note: The image-exif-data 427 function and the Exif metadata's attributes can be used to supply the input strings.

Examples

- **altova:format-geolocation**(33.33, -22.22, 4) returns the xs:string "33.33 -22.22"
- **altova:format-geolocation**(33.33, -22.22, 2) returns the xs:string "33.33N 22.22W"
- **altova:format-geolocation**(-33.33, 22.22, 2) returns the xs:string "33.33S 22.22E"
- **altova:format-geolocation**(33.33, -22.22, 1) returns the xs:string "33°19'48.00"S 22° 13'12.00"E"

Geolocation output string formats:

The supplied latitude and longitude is formatted in one of the output formats given below. The desired format is identified by its integer ID (`1 to 4`). Latitude values range from `+90` to `-90` (`N` to `S`). Longitude values range from `+180` to `-180` (`E` to `W`).

1
Degrees, minutes, decimal seconds, with suffixed orientation (`N/S, E/W`)

© 2017-2023 Altova GmbH

`D°M'S.SS"N/S D°M'S.SS"E/W` *Example*: `33°55'11.11"N 22°44'66.66"W`

2
Decimal degrees, with suffixed orientation (`N/S`, `E/W`) `D.DDN/S D.DDE/W` *Example*: `33.33N 22.22W`

3
Degrees, minutes, decimal seconds, with prefixed sign (`+/-`); plus sign for (`N/E`) is optional `+/-D°M'S.SS" +/-D°M'S.SS"` *Example*: `33°55'11.11" -22°44'66.66"`

4
Decimal degrees, with prefixed sign (`+/-`); plus sign for (`N/E`) is optional `+/-D.DD +/-D.DD` *Example*: `33.33 -22.22`

Altova Exif Attribute: Geolocation

The Altova XPath/XQuery Engine generates the custom attribute `Geolocation` from standard Exif metadata tags. `Geolocation` is a concatenation of four Exif tags: `GPSLatitude`, `GPSLatitudeRef`, `GPSLongitude`, `GPSLongitudeRef`, with units added (*see table below*).

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151° 13'11.73"E

parse-geolocation [altova:]

`altova:parse-geolocation(GeolocationInputString` *`as xs:string`*`)` **`as xs:decimal+`** **XP3.1** **XQ3.1**

Parses the supplied `GeolocationInputString` argument and returns the geolocation's latitude and longitude (in that order) as a sequence two `xs:decimal` items. The formats in which the geolocation input string can be supplied are listed below.

Note: The `image-exif-data` 427 function and the Exif metadata's `@Geolocation` 427 attribute can be used to supply the geolocation input string (*see example below*).

Examples

- `altova:parse-geolocation("33.33 -22.22")` returns the sequence of two `xs:decimals` `(33.33, 22.22)`

- `altova:parse-geolocation("48°51'29.6""N  24°17'40.2""")` returns the sequence of two `xs:decimals (48.8582222222222, 24.2945)`
- `altova:parse-geolocation('48°51''29.6"N  24°17''40.2"')` returns the sequence of two `xs:decimals (48.8582222222222, 24.2945)`
- `altova:parse-geolocation( image-exif-data(//MyImages/Image20141130.01)/@Geolocation )` returns a sequence of two `xs:decimals`

Geolocation input string formats:

The geolocation input string must contain latitude and longitude (in that order) separated by whitespace. Each can be in any of the following formats. Combinations are allowed. So latitude can be in one format and longitude can be in another. Latitude values range from `+90` to `-90` (`N` to `S`). Longitude values range from `+180` to `-180` (`E` to `W`).

Note: If single quotes or double quotes are used to delimit the input string argument, this will create a mismatch with the single quotes or double quotes that are used, respectively, to indicate minute-values and second-values. In such cases, the quotes that are used for indicating minute-values and second-values must be escaped by doubling them. In the examples in this section, quotes used to delimit the input string are highlighted in yellow (`"`) while unit indicators that are escaped are highlighted in blue (`""`).

- Degrees, minutes, decimal seconds, with suffixed orientation (`N/S`, `E/W`)
  `D°M'S.SS"N/S  D°M'S.SS"W/E`
  Example: **`33°55'11.11"N  22°44'55.25"W`**

- Degrees, minutes, decimal seconds, with prefixed sign (`+/-`); the plus sign for (**`N/E`**) is optional
  `+/-D°M'S.SS"  +/-D°M'S.SS"`
  Example: **`33°55'11.11"  -22°44'55.25"`**

- Degrees, decimal minutes, with suffixed orientation (`N/S`, `E/W`)
  `D°M.MM'N/S  D°M.MM'W/E`
  Example: **`33°55.55'N  22°44.44'W`**

- Degrees, decimal minutes, with prefixed sign (`+/-`); the plus sign for (**`N/E`**) is optional
  `+/-D°M.MM'  +/-D°M.MM'`
  Example: **`+33°55.55'  -22°44.44'`**

- Decimal degrees, with suffixed orientation (`N/S`, `E/W`)
  `D.DDN/S  D.DDW/E`
  Example: **`33.33N  22.22W`**

- Decimal degrees, with prefixed sign (`+/-`); the plus sign for (**`N/S E/W`**) is optional
  `+/-D.DD  +/-D.DD`
  Example: **`33.33  -22.22`**

Examples of format-combinations:

```
33.33N  -22°44'55.25"
33.33  22°44'55.25"W
33.33  22.45
```


© 2017-2023 Altova GmbH

Altova Exif Attribute: Geolocation

The Altova XPath/XQuery Engine generates the custom attribute **Geolocation** from standard Exif metadata tags. **Geolocation** is a concatenation of four Exif tags: GPSLatitude, GPSLatitudeRef, GPSLongitude, GPSLongitudeRef, with units added (*see table below*).

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151° 13'11.73"E

geolocation-distance-km [altova:]

altova:geolocation-distance-km(GeolocationInputString-1 *as xs:string*, **GeolocationInputString-2** *as xs:string*) **as xs:decimal** XP3.1 XQ3.1

Calculates the distance between two geolocations in kilometers. The formats in which the geolocation input string can be supplied are listed below. Latitude values range from +90 to -90 (N to S). Longitude values range from +180 to -180 (E to W).

Note: The image-exif-data[427] function and the Exif metadata's @Geolocation[427] attribute can be used to supply geolocation input strings.

Examples

- **altova:geolocation-distance-km**("33.33 -22.22", "48°51'29.6""N 24°17'40.2""") returns the xs:decimal 4183.08132372392

Geolocation input string formats:

The geolocation input string must contain latitude and longitude (in that order) separated by whitespace. Each can be in any of the following formats. Combinations are allowed. So latitude can be in one format and longitude can be in another. Latitude values range from +90 to -90 (N to S). Longitude values range from +180 to -180 (E to W).

Note: If single quotes or double quotes are used to delimit the input string argument, this will create a mismatch with the single quotes or double quotes that are used, respectively, to indicate minute-values and second-values. In such cases, the quotes that are used for indicating minute-values and second-values must be escaped by doubling them. In the examples in this section, quotes used to delimit the input string are highlighted in yellow (") while unit indicators that are escaped are highlighted in blue ("").

- Degrees, minutes, decimal seconds, with suffixed orientation (N/S, E/W)
  D°M'S.SS"N/S D°M'S.SS"W/E
  Example: **33°55'11.11"N 22°44'55.25"W**

- Degrees, minutes, decimal seconds, with prefixed sign (+/-); the plus sign for (**N/E**) is optional
  +/-D°M'S.SS" +/-D°M'S.SS"
  Example: **33°55'11.11" -22°44'55.25"**

- Degrees, decimal minutes, with suffixed orientation (N/S, E/W)
  D°M.MM'N/S D°M.MM'W/E
  Example: **33°55.55'N 22°44.44'W**

- Degrees, decimal minutes, with prefixed sign (+/-); the plus sign for (**N/E**) is optional
  +/-D°M.MM' +/-D°M.MM'
  Example: **+33°55.55' -22°44.44'**

- Decimal degrees, with suffixed orientation (N/S, E/W)
  D.DDN/S D.DDW/E
  Example: **33.33N 22.22W**

- Decimal degrees, with prefixed sign (+/-); the plus sign for (**N/S E/W**) is optional
  +/-D.DD +/-D.DD
  Example: **33.33 -22.22**

Examples of format-combinations:
33.33N -22°44'55.25"
33.33 22°44'55.25"W
33.33 22.45

Altova Exif Attribute: Geolocation

The Altova XPath/XQuery Engine generates the custom attribute **Geolocation** from standard Exif metadata tags. **Geolocation** is a concatenation of four Exif tags: GPSLatitude, GPSLatitudeRef, GPSLongitude, GPSLongitudeRef, with units added (*see table below*).

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151° 13'11.73"E

geolocation-distance-mi [altova:]

altova:geolocation-distance-mi(GeolocationInputString-1 *as xs:string*, **GeolocationInputString-2** *as xs:string*) **as xs:decimal** **XP3.1** **XQ3.1**
Calculates the distance between two geolocations in miles. The formats in which a geolocation input string can be supplied are listed below. Latitude values range from +90 to -90 (N to S). Longitude values range from +180 to -180 (E to W).

Note: The image-exif-data 427 function and the Exif metadata's @Geolocation 427 attribute can be used to supply geolocation input strings.

Examples

- **altova:geolocation-distance-mi**("33.33 -22.22", "48°51'29.6""N 24°17'40.2""")
  returns the xs:decimal 2599.40652340653

© 2017-2023 Altova GmbH

Geolocation input string formats:

The geolocation input string must contain latitude and longitude (in that order) separated by whitespace. Each can be in any of the following formats. Combinations are allowed. So latitude can be in one format and longitude can be in another. Latitude values range from `+90` to `-90` (`N` to `S`). Longitude values range from `+180` to `-180` (`E` to `W`).

Note: If single quotes or double quotes are used to delimit the input string argument, this will create a mismatch with the single quotes or double quotes that are used, respectively, to indicate minute-values and second-values. In such cases, the quotes that are used for indicating minute-values and second-values must be escaped by doubling them. In the examples in this section, quotes used to delimit the input string are highlighted in yellow (`"`) while unit indicators that are escaped are highlighted in blue (`""`).

- Degrees, minutes, decimal seconds, with suffixed orientation (`N/S`, `E/W`)
  `D°M'S.SS"N/S D°M'S.SS"W/E`
  Example: **`33°55'11.11"N 22°44'55.25"W`**

- Degrees, minutes, decimal seconds, with prefixed sign (`+/-`); the plus sign for (**`N/E`**) is optional
  `+/-D°M'S.SS" +/-D°M'S.SS"`
  Example: **`33°55'11.11" -22°44'55.25"`**

- Degrees, decimal minutes, with suffixed orientation (`N/S`, `E/W`)
  `D°M.MM'N/S D°M.MM'W/E`
  Example: **`33°55.55'N 22°44.44'W`**

- Degrees, decimal minutes, with prefixed sign (`+/-`); the plus sign for (**`N/E`**) is optional
  `+/-D°M.MM' +/-D°M.MM'`
  Example: **`+33°55.55' -22°44.44'`**

- Decimal degrees, with suffixed orientation (`N/S`, `E/W`)
  `D.DDN/S D.DDW/E`
  Example: **`33.33N 22.22W`**

- Decimal degrees, with prefixed sign (`+/-`); the plus sign for (**`N/S E/W`**) is optional
  `+/-D.DD +/-D.DD`
  Example: **`33.33 -22.22`**

Examples of format-combinations:

```
33.33N  -22°44'55.25"
33.33  22°44'55.25"W
33.33  22.45
```

Altova Exif Attribute: Geolocation

The Altova XPath/XQuery Engine generates the custom attribute **`Geolocation`** from standard Exif metadata tags. **`Geolocation`** is a concatenation of four Exif tags: `GPSLatitude`, `GPSLatitudeRef`, `GPSLongitude`, `GPSLongitudeRef`, with units added (*see table below*).

GPSLatitude	GPSLatitudeRe	GPSLongitude	GPSLongitudeRe	Geolocation

	f		f	
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151° 13'11.73"E

geolocations-bounding-rectangle [altova:]

altova:geolocations-bounding-rectangle(Geolocations *as xs:sequence*, **GeolocationOutputStringFormat** *as xs:integer*) **as xs:string** XP3.1 XQ3.1

Takes a sequence of strings as its first argument; each string in the sequence is a geolocation. The function returns a sequence of two strings which are, respectively, the top-left and bottom-right geolocation coordinates of a bounding rectangle that is optimally sized to enclose all the geolocations submitted in the first argument. The formats in which a geolocation input string can be supplied are listed below (*see 'Geolocation input string formats'*). Latitude values range from `+90` to `-90` (`N` to `S`). Longitude values range from `+180` to `-180` (`E` to `W`).

The function's second argument specifies the format of the two geolocation strings in the output sequence. The argument takes an integer value from `1` to `4`, where each value identifies a different geolocation string format (*see 'Geolocation output string formats' below*).

Note: The image-exif-data 427 function and the Exif metadata's attributes can be used to supply the input strings.

Examples

- **altova:geolocations-bounding-rectangle**(("48.2143531 16.3707266", "51.50939 -0.11832"), 1) returns the sequence ("51°30'33.804"N 0°7'5.952"W", "48°12'51.67116"N 16°22'14.61576"E")
- **altova:geolocations-bounding-rectangle**(("48.2143531 16.3707266", "51.50939 -0.11832", "42.5584577 -70.8893334"), 4) returns the sequence ("51.50939 -70.8893334", "42.5584577 16.3707266")

Geolocation input string formats:

The geolocation input string must contain latitude and longitude (in that order) separated by whitespace. Each can be in any of the following formats. Combinations are allowed. So latitude can be in one format and longitude can be in another. Latitude values range from `+90` to `-90` (`N` to `S`). Longitude values range from `+180` to `-180` (`E` to `W`).

Note: If single quotes or double quotes are used to delimit the input string argument, this will create a mismatch with the single quotes or double quotes that are used, respectively, to indicate minute-values and second-values. In such cases, the quotes that are used for indicating minute-values and second-values must be escaped by doubling them. In the examples in this section, quotes used to delimit the input string are highlighted in yellow (") while unit indicators that are escaped are highlighted in blue ("").

- Degrees, minutes, decimal seconds, with suffixed orientation (`N/S`, `E/W`)
  D°M'S.SS"N/S D°M'S.SS"W/E
  Example: **33°55'11.11"N 22°44'55.25"W**

© 2017-2023 Altova GmbH

- Degrees, minutes, decimal seconds, with prefixed sign (+/-); the plus sign for (N/E) is optional
  +/-D°M'S.SS" +/-D°M'S.SS"
  Example: **33°55'11.11" -22°44'55.25"**

- Degrees, decimal minutes, with suffixed orientation (N/S, E/W)
  D°M.MM'N/S D°M.MM'W/E
  Example: **33°55.55'N 22°44.44'W**

- Degrees, decimal minutes, with prefixed sign (+/-); the plus sign for (N/E) is optional
  +/-D°M.MM' +/-D°M.MM'
  Example: **+33°55.55' -22°44.44'**

- Decimal degrees, with suffixed orientation (N/S, E/W)
  D.DDN/S D.DDW/E
  Example: **33.33N 22.22W**

- Decimal degrees, with prefixed sign (+/-); the plus sign for (N/S E/W) is optional
  +/-D.DD +/-D.DD
  Example: **33.33 -22.22**

Examples of format-combinations:
33.33N -22°44'55.25"
33.33 22°44'55.25"W
33.33 22.45

Geolocation output string formats:

The supplied latitude and longitude is formatted in one of the output formats given below. The desired format is identified by its integer ID (1 to 4). Latitude values range from +90 to -90 (N to S). Longitude values range from +180 to -180 (E to W).

1
Degrees, minutes, decimal seconds, with suffixed orientation (N/S, E/W) **D°M'S.SS"N/S D°M'S.SS"E/W** *Example:* **33°55'11.11"N 22°44'66.66"W**

2
Decimal degrees, with suffixed orientation (N/S, E/W) **D.DDN/S D.DDE/W** *Example:* **33.33N 22.22W**

3
Degrees, minutes, decimal seconds, with prefixed sign (+/-); plus sign for (N/E) is optional **+/-D°M'S.SS" +/-D°M'S.SS"** *Example:* **33°55'11.11" -22°44'66.66"**

4
Decimal degrees, with prefixed sign (+/-); plus sign for (N/E) is optional +/-D.DD +/-D.DD *Example*: 33.33 -22.22

- *Altova Exif Attribute: Geolocation*

  The Altova XPath/XQuery Engine generates the custom attribute **Geolocation** from standard Exif metadata tags. **Geolocation** is a concatenation of four Exif tags: GPSLatitude, GPSLatitudeRef, GPSLongitude, GPSLongitudeRef, with units added (*see table below*).

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151° 13'11.73"E

- geolocation-within-polygon [altova:]

  altova:geolocation-within-polygon(Geolocation *as xs:string*, **((PolygonPoint** *as xs:string*)+)) **as xs:boolean** XP3.1 XQ3.1

  Determines whether **Geolocation** (the first argument) is within the polygonal area described by the **PolygonPoint** arguments. If the PolygonPoint arguments do not form a closed figure (formed when the first point and the last point are the same), then the first point is implicitly added as the last point in order to close the figure. All the arguments (Geolocation and PolygonPoint+) are given by geolocation input strings (*formats listed below*). If the Geolocation argument is within the polygonal area, then the function returns true(); otherwise it returns false(). Latitude values range from +90 to -90 (N to S). Longitude values range from +180 to -180 (E to W).

  Note: The image-exif-data[427] function and the Exif metadata's @Geolocation[427] attribute can be used to supply geolocation input strings.

  - *Examples*

    - **altova:geolocation-within-polygon**("33 -22", ("58 -32", "-78 -55", "48 24", "58 -32")) returns true()
    - **altova:geolocation-within-polygon**("33 -22", ("58 -32", "-78 -55", "48 24")) returns true()
    - **altova:geolocation-within-polygon**("33 -22", ("58 -32", "-78 -55", "48°51'29.6""N 24°17'40.2""")) returns true()

  - *Geolocation input string formats:*

    The geolocation input string must contain latitude and longitude (in that order) separated by whitespace. Each can be in any of the following formats. Combinations are allowed. So latitude can be in one format and longitude can be in another. Latitude values range from +90 to -90 (N to S). Longitude values range from +180 to -180 (E to W).

© 2017-2023 Altova GmbH

Note: If single quotes or double quotes are used to delimit the input string argument, this will create a mismatch with the single quotes or double quotes that are used, respectively, to indicate minute-values and second-values. In such cases, the quotes that are used for indicating minute-values and second-values must be escaped by doubling them. In the examples in this section, quotes used to delimit the input string are highlighted in yellow (") while unit indicators that are escaped are highlighted in blue ("").

- Degrees, minutes, decimal seconds, with suffixed orientation (N/S, E/W)
  D°M'S.SS"N/S D°M'S.SS"W/E
  Example: **33°55'11.11"N 22°44'55.25"W**

- Degrees, minutes, decimal seconds, with prefixed sign (+/-); the plus sign for (**N/E**) is optional
  +/-D°M'S.SS" +/-D°M'S.SS"
  Example: **33°55'11.11" -22°44'55.25"**

- Degrees, decimal minutes, with suffixed orientation (N/S, E/W)
  D°M.MM'N/S D°M.MM'W/E
  Example: **33°55.55'N 22°44.44'W**

- Degrees, decimal minutes, with prefixed sign (+/-); the plus sign for (**N/E**) is optional
  +/-D°M.MM' +/-D°M.MM'
  Example: **+33°55.55' -22°44.44'**

- Decimal degrees, with suffixed orientation (N/S, E/W)
  D.DDN/S D.DDW/E
  Example: **33.33N 22.22W**

- Decimal degrees, with prefixed sign (+/-); the plus sign for (**N/S E/W**) is optional
  +/-D.DD +/-D.DD
  Example: **33.33 -22.22**

Examples of format-combinations:
33.33N -22°44'55.25"
33.33 22°44'55.25"W
33.33 22.45

Altova Exif Attribute: Geolocation

The Altova XPath/XQuery Engine generates the custom attribute **Geolocation** from standard Exif metadata tags. **Geolocation** is a concatenation of four Exif tags: GPSLatitude, GPSLatitudeRef, GPSLongitude, GPSLongitudeRef, with units added (*see table below*).

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151° 13'11.73"E

geolocation-within-rectangle [altova:]

altova:geolocation-within-rectangle(Geolocation *as xs:string*, **RectCorner-1** *as xs:string*, **RectCorner-2** *as xs:string*) **as xs:boolean** **XP3.1** **XQ3.1**

Determines whether **Geolocation** (the first argument) is within the rectangle defined by the second and third arguments, **RectCorner-1** and **RectCorner-2**, which specify opposite corners of the rectangle. All the arguments (`Geolocation`, **RectCorner-1** and **RectCorner-2**) are given by geolocation input strings (*formats listed below*). If the `Geolocation` argument is within the rectangle, then the function returns `true()`; otherwise it returns `false()`. Latitude values range from `+90` to `-90` (`N` to `S`). Longitude values range from `+180` to `-180` (`E` to `W`).

Note: The image-exif-data 427 function and the Exif metadata's @Geolocation 427 attribute can be used to supply geolocation input strings.

Examples

- **altova:geolocation-within-rectangle(**"33 -22", "58 -32", "-48 24") returns `true()`
- **altova:geolocation-within-rectangle(**"33 -22", "58 -32", "48 24") returns `false()`
- **altova:geolocation-within-rectangle(**"33 -22", "58 -32", "48°51'29.6""S 24° 17'40.2""") returns `true()`

Geolocation input string formats:

The geolocation input string must contain latitude and longitude (in that order) separated by whitespace. Each can be in any of the following formats. Combinations are allowed. So latitude can be in one format and longitude can be in another. Latitude values range from `+90` to `-90` (`N` to `S`). Longitude values range from `+180` to `-180` (`E` to `W`).

Note: If single quotes or double quotes are used to delimit the input string argument, this will create a mismatch with the single quotes or double quotes that are used, respectively, to indicate minute-values and second-values. In such cases, the quotes that are used for indicating minute-values and second-values must be escaped by doubling them. In the examples in this section, quotes used to delimit the input string are highlighted in yellow (") while unit indicators that are escaped are highlighted in blue ("").

- Degrees, minutes, decimal seconds, with suffixed orientation (`N/S`, `E/W`)
  D°M'S.SS"N/S D°M'S.SS"W/E
  Example: **33°55'11.11"N 22°44'55.25"W**

- Degrees, minutes, decimal seconds, with prefixed sign (`+/-`); the plus sign for (**N/E**) is optional
  +/-D°M'S.SS" +/-D°M'S.SS"
  Example: **33°55'11.11" -22°44'55.25"**

- Degrees, decimal minutes, with suffixed orientation (`N/S`, `E/W`)
  D°M.MM'N/S D°M.MM'W/E
  Example: **33°55.55'N 22°44.44'W**

- Degrees, decimal minutes, with prefixed sign (`+/-`); the plus sign for (**N/E**) is optional
  +/-D°M.MM' +/-D°M.MM'
  Example: **+33°55.55' -22°44.44'**

© 2017-2023 Altova GmbH

- Decimal degrees, with suffixed orientation (N/S, E/W)
  D.DDN/S D.DDW/E
  Example: **33.33N 22.22W**

- Decimal degrees, with prefixed sign (+/-); the plus sign for (**N/S E/W**) is optional
  +/-D.DD +/-D.DD
  Example: **33.33 -22.22**

Examples of format-combinations:
33.33N -22°44'55.25"
33.33 22°44'55.25"W
33.33 22.45

Altova Exif Attribute: Geolocation

The Altova XPath/XQuery Engine generates the custom attribute **Geolocation** from standard Exif metadata tags. **Geolocation** is a concatenation of four Exif tags: GPSLatitude, GPSLatitudeRef, GPSLongitude, GPSLongitudeRef, with units added (*see table below*).

GPSLatitude	GPSLatitudeRef	GPSLongitude	GPSLongitudeRef	Geolocation
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151° 13'11.73"E

[Top 416]

10.2.1.4 XPath/XQuery Functions: Image-Related

The following image-related XPath/XQuery extension functions are supported in the current version of RaptorXML Server 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 XP3.1
XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3

XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3.1

▼ suggested-image-file-extension [altova:]

altova:suggested-image-file-extension(Base64String *as string***) as string? XP3.1 XQ3.1**
Takes the Base64 encoding of an image file as its argument and returns the file extension of the image as recorded in the Base64-encoding of the image. The returned value is a suggestion based on the image type information available in the encoding. If this information is not available, then an empty string is returned. This function is useful if you wish to save a Base64 image as a file and wish to dynamically retrieve an appropriate file extension.

Examples

- `altova:suggested-image-file-extension(/MyImages/MobilePhone/Image20141130.01)` returns `'jpg'`
- `altova:suggested-image-file-extension($XML1/Staff/Person/@photo)` returns `''`

In the examples above, the nodes supplied as the argument of the function are assumed to contain a Base64-encoded image. The first example retrieves `jpg` as the file's type and extension. In the second example, the submitted Base64 encoding does not provide usable file extension information.

▼ image-exif-data [altova:]

altova:image-exif-data(Base64BinaryString *as string***) as element? XP3.1 XQ3.1**
Takes a Base64-encoded JPEG image as its argument and returns an element called **Exif** that contains the Exif metadata of the image. The Exif metadata is created as attribute-value pairs of the `Exif` element. The attribute names are the Exif data tags found in the Base64 encoding. The list of Exif-specification tags is given below. If a vendor-specific tag is present in the Exif data, this tag and its value will also be returned as an attribute-value pair. Additional to the standard Exif metadata tags (*see list below*), Altova-specific attribute-value pairs are also generated. These Altova Exif attributes are listed below.

Examples

- To access any one attribute, use the function like this:
  `image-exif-data(//MyImages/Image20141130.01)/@GPSLatitude`
  `image-exif-data(//MyImages/Image20141130.01)/@Geolocation`
- To access all the attributes, use the function like this:
  `image-exif-data(//MyImages/Image20141130.01)/@*`
- To access the names of all the attributes, use the following expression:
  `for $i in image-exif-data(//MyImages/Image20141130.01)/@* return name($i)`
  This is useful to find out the names of the attributes returned by the function.

Altova Exif Attribute: Geolocation

The Altova XPath/XQuery Engine generates the custom attribute **Geolocation** from standard Exif metadata tags. **Geolocation** is a concatenation of four Exif tags: `GPSLatitude`, `GPSLatitudeRef`, `GPSLongitude`, `GPSLongitudeRef`, with units added (*see table below*).

GPSLatitude	GPSLatitudeRe	GPSLongitude	GPSLongitudeRe	Geolocation

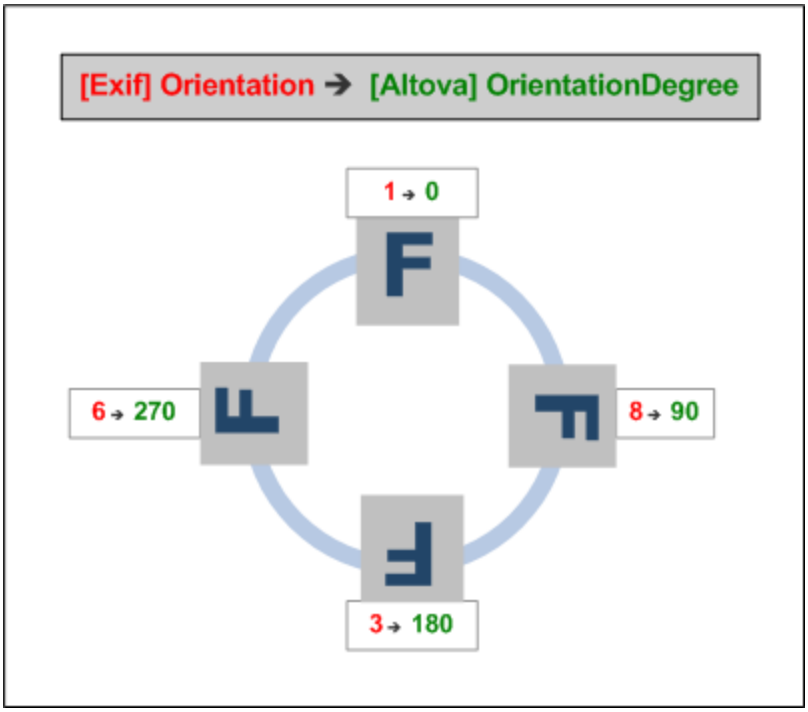
© 2017-2023 Altova GmbH

	f		f	
33 51 21.91	S	151 13 11.73	E	33°51'21.91"S 151° 13'11.73"E

- *Altova Exif Attribute: OrientationDegree*

The Altova XPath/XQuery Engine generates the custom attribute **OrientationDegree** from the Exif metadata tag **Orientation**.

OrientationDegree translates the standard Exif tag Orientation from an integer value (1, 8, 3, or 6) to the respective degree values of each (0, 90, 180, 270), as shown in the figure below. Note that there are no translations of the Orientation values of 2, 4, 5, 7. (These orientations are obtained by flipping image 1 across its vertical center axis to get the image with a value of 2, and then rotating this image in 90-degree jumps clockwise to get the values of 7, 4, and 5, respectively).



- *Listing of standard Exif meta tags*

- ImageWidth
- ImageLength
- BitsPerSample
- Compression
- PhotometricInterpretation
- Orientation
- SamplesPerPixel
- PlanarConfiguration
- YCbCrSubSampling

- YCbCrPositioning
- XResolution
- YResolution
- ResolutionUnit
- StripOffsets
- RowsPerStrip
- StripByteCounts
- JPEGInterchangeFormat
- JPEGInterchangeFormatLength
- TransferFunction
- WhitePoint
- PrimaryChromaticities
- YCbCrCoefficients
- ReferenceBlackWhite
- DateTime
- ImageDescription
- Make
- Model
- Software
- Artist
- Copyright

- ExifVersion
- FlashpixVersion
- ColorSpace
- ComponentsConfiguration
- CompressedBitsPerPixel
- PixelXDimension
- PixelYDimension
- MakerNote
- UserComment
- RelatedSoundFile
- DateTimeOriginal
- DateTimeDigitized
- SubSecTime
- SubSecTimeOriginal
- SubSecTimeDigitized
- ExposureTime
- FNumber
- ExposureProgram
- SpectralSensitivity
- ISOSpeedRatings
- OECF
- ShutterSpeedValue
- ApertureValue
- BrightnessValue
- ExposureBiasValue
- MaxApertureValue
- SubjectDistance
- MeteringMode
- LightSource
- Flash
- FocalLength
- SubjectArea

 © 2017-2023 Altova GmbH

- FlashEnergy
- SpatialFrequencyResponse
- FocalPlaneXResolution
- FocalPlaneYResolution
- FocalPlaneResolutionUnit
- SubjectLocation
- ExposureIndex
- SensingMethod
- FileSource
- SceneType
- CFAPattern
- CustomRendered
- ExposureMode
- WhiteBalance
- DigitalZoomRatio
- FocalLengthIn35mmFilm
- SceneCaptureType
- GainControl
- Contrast
- Saturation
- Sharpness
- DeviceSettingDescription
- SubjectDistanceRange
- ImageUniqueID

- GPSVersionID
- GPSLatitudeRef
- GPSLatitude
- GPSLongitudeRef
- GPSLongitude
- GPSAltitudeRef
- GPSAltitude
- GPSTimeStamp
- GPSSatellites
- GPSStatus
- GPSMeasureMode
- GPSDOP
- GPSSpeedRef
- GPSSpeed
- GPSTrackRef
- GPSTrack
- GPSImgDirectionRef
- GPSImgDirection
- GPSMapDatum
- GPSDestLatitudeRef
- GPSDestLatitude
- GPSDestLongitudeRef
- GPSDestLongitude
- GPSDestBearingRef
- GPSDestBearing
- GPSDestDistanceRef
- GPSDestDistance
- GPSProcessingMethod
- GPSAreaInformation

- GPSDateStamp
- GPSDifferential

[Top 427]

10.2.1.5 XPath/XQuery Functions: Numeric

Altova's numeric extension functions can be used in XPath and XQuery expressions and provide additional functionality for the processing of data. 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 XP3.1
XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3
XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3.1

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 XP3.1 XQ3.1

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

 © 2017-2023 Altova GmbH

`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** **XP3.1** **XQ3.1**
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 432]

Numeric functions

▼ hex-string-to-integer [altova:]

`altova:hex-string-to-integer(HexString` *`as xs:string`***`) as xs:integer`** **XP3.1** **XQ3.1**
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`** **XP3.1** **XQ3.1**
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 432]

Number-formatting functions

[Top 432]

10.2.1.6 XPath/XQuery Functions: Schema

The Altova extension functions listed below return schema information. Given below are descriptions of the functions, together with (i) examples and (ii) a listing of schema components and their respective properties. They can be used with Altova's **XPath 3.0** and **XQuery 3.0** engines and are available in XPath/XQuery contexts.

Schema information from schema documents

The function `altova:schema` has two arguments: one with zero arguments and the other with two arguments. The zero-argument function returns the whole schema. You can then, from this starting point, navigate into the schema to locate the schema components you want. The two-argument function returns a specific component kind that is identified by its QName. In both cases, the return value is a function. To navigate into the returned component, you must select a property of that specific component. If the property is a non-atomic item (that is, if it is a component), then you can navigate further by selecting a property of this component. If the selected property is an atomic item, then the value of the item is returned and you cannot navigate any further.

Note: In XPath expressions, the schema must be imported into the processing environment (for example, into XSLT) with the `xslt:import-schema` instruction. In XQuery expressions, the schema must be explicitly imported using a schema import.

Schema information from XML nodes

The function `altova:type` submits the node of an XML document and returns the node's type information from the PSVI.

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.

 © 2017-2023 Altova GmbH

XPath functions (used in XPath expressions in XSLT):	XP1 XP2 XP3.1
XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3
XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3.1

▼ Schema (zero arguments)

altova:schema() as (function(xs:string) as item()*)? XP3.1 XQ3.1
Returns the **schema** component as a whole. You can navigate further into the **schema** component by selecting one of the **schema** component's properties.

- If this property is a component, you can navigate another step deeper by selecting one of this component's properties. This step can be repeated to navigate further into the schema.
- If the component is an atomic value, the atomic value is returned and you cannot navigate any deeper.

The properties of the **schema** component are:

```
"type definitions"
"attribute declarations"
"element declarations"
"attribute group definitions"
"model group definitions"
"notation declarations"
"identity-constraint definitions"
```

The properties of all other component kinds (besides **schema**) are listed below.

Note: In XQuery expressions, the schema must be explicitly imported. In XPath expressions, the schema must have been imported into the processing environment, for example, into XSLT with the **xslt:import** instruction.

▣ *Examples*

- `import schema "" at "C:\Test\ExpReport.xsd"; for $typedef in altova:schema() ("type definitions") return $typedef ("name")` returns the names of all simple types or complex types in the schema

- `import schema "" at "C:\Test\ExpReport.xsd"; altova:schema() ("type definitions")[1]("name")` returns the name of the first of all simple types or complex types in the schema

Components and their properties

▣ Assertion

Property name	Property type	Property value
kind	string	"Assertion"

test	XPath Property Record	

Attribute Declaration

Property name	Property type	Property value
kind	string	"Attribute Declaration"
name	string	Local name of the attribute
target namespace	string	Namespace URI of the attribute
type definition	Simple Type or Complex Type	
scope	A function with properties ("class":"Scope", "variety": "global" or "local", "parent": the containing Complex Type or Attribute Group)	
value constraint	If present, a function with properties ("class": "Value Constraint", "variety": "fixed" or "default", "value": atomic value, "lexical form": string. Note that the "value" property is not available for namespace-sensitive types	
inheritable	boolean	

Attribute Group Declaration

Property name	Property type	Property value
kind	string	"Attribute Group Definition"
name	string	Local name of the attribute group
target namespace	string	Namespace URI of the attribute group
attribute uses	Sequence of (Attribute Use)	
attribute wildcard	Optional Attribute Wildcard	

Attribute Use

Property name	Property type	Property value
kind	string	"Attribute Use"
required	boolean	true if the attribute is required, false if optional
value constraint	See Attribute Declaration	
inheritable	boolean	

Attribute Wildcard

Property name	Property type	Property value

 © 2017-2023 Altova GmbH

kind	string	"Wildcard"
namespace constraint	function with properties ("class": "Namespace Constraint", "variety": "any"|"enumeration"|"not", "namespaces": sequence of xs:anyURI, "disallowed names": list containing QNames and/or the strings "defined" and "definedSiblings"	
process contents	string ("strict"|"lax"|"skip")	

Complex Type

Property name	Property type	Property value
kind	string	"Complex Type"
name	string	Local name of the type (empty if anonymous)
target namespace	string	Namespace URI of the type (empty if anonymous)
base type definition	Complex Type Definition	
final	Sequence of strings ("restriction"|"extension")	
context	Empty sequence (not implemented)	
derivation method	string ("restriction"|"extension")	
abstract	boolean	
attribute uses	Sequence of Attribute Use	
attribute wildcard	Optional Attribute Wildcard	
content type	function with properties: ("class":"Content Type", "variety":string ("element-only"|"empty"|"mixed"|"simple"), particle: optional Particle, "open content": function with properties ("class":"Open Content", "mode": string ("interleave"|"suffix"), "wildcard": Wildcard), "simple type definition": Simple Type)	
prohibited substitutions	Sequence of strings ("restriction"|"extension")	
assertions	Sequence of Assertion	

Element Declaration

Property name	Property type	Property value

kind	string	"Complex Type"
name	string	Local name of the type (empty if anonymous)
target namespace	string	Namespace URI of the type (empty if anonymous)
type definition	Simple Type or Complex Type	
type table	function with properties ("class":"Type Table", "alternatives": sequence of Type Alternative, "default type definition": Simple Type or Complex Type)	
scope	function with properties ("class":"Scope", "variety": ("global"|"local"), "parent": optional Complex Type)	
value constraint	see Attribute Declaration	
nillable	boolean	
identity-constraint definitions	Sequence of Identity Constraint	
substitution group affiliations	Sequence of Element Declaration	
substitution group exclusions	Sequence of strings ("restriction"|"extension")	
disallowed substitutions	Sequence of strings ("restriction"|"extension"|"substitution")	
abstract	boolean	

Element Wildcard

Property name	Property type	Property value
kind	string	"Wildcard"
namespace constraint	function with properties ("class": "Namespace Constraint", "variety": "any"|"enumeration"|"not", "namespaces": sequence of xs:anyURI, "disallowed names": list containing QNames and/or the strings "defined" and "definedSiblings"	
process contents	string ("strict"|"lax"|"skip")	

Facet

Property name	Property type	Property value
kind	string	The name of the facet, for example "minLength" or

© 2017-2023 Altova GmbH

		"enumeration"
value	depends on facet	The value of the facet
fixed	boolean	
typed-value	For the enumeration facet only, array(xs:anyAtomicType*)	An array containing the enumeration values, each of which may in general be a sequence of atomic values. (Note: for the enumeration facet, the "value" property is a sequence of strings, regardless of the actual type)

Identity Constraint

Property name	Property type	Property value
kind	string	"Identity-Constraint Definition"
name	string	Local name of the constraint
target namespace	string	Namespace URI of the constraint
identity-constraint category	string ("key"|"unique"|"keyRef")	
selector	XPath Property Record	
fields	Sequence of XPath Property Record	
referenced key	(For keyRef only): Identity Constraint	The corresponding key constraint

Model Group

Property name	Property type	Property value
kind	string	"Model Group"
compositor	string ("sequence"|"choice"|"all")	
particles	Sequence of Particle	

Model Group Definition

Property name	Property type	Property value
kind	string	"Model Group Definition"
name	string	Local name of the model group
target namespace	string	Namespace URI of the model group
model group	Model Group	

Notation

Property name	Property type	Property value

kind	string	"Notation Declaration"
name	string	Local name of the notation
target namespace	string	Namespace URI of the notation
system identifier	anyURI	
public identifier	string	

Particle

Property name	Property type	Property value
kind	string	"Particle"
min occurs	integer	
max occurs	integer, or string("unbounded")	
term	Element Declaration, Element Wildcard, or ModelGroup	

Simple Type

Property name	Property type	Property value
kind	string	"Simple Type Definition"
name	string	Local name of the type (empty if anonymous)
target namespace	string	Namespace URI of the type (empty if anonymous)
final	Sequence of string("restriction"|"extension"|"list"|"union")	
context	containing component	
base type definition	Simple Type	
facets	Sequence of Facet	
fundamental facets	Empty sequence (not implemented)	
variety	string ("atomic"|"list"|"union")	
primitive type definition	Simple Type	
item type definition	(for list types only) Simple Type	
member type definitions	(for union types only) Sequence of Simple Type	

Type Alternative

Property name	Property type	Property value
kind	string	"Type Alternative"

© 2017-2023 Altova GmbH

test	XPath Property Record	
type definition	Simple Type or Complex Type	

XPath Property Record

Property name	Property type	Property value
namespace bindings	Sequence of functions with properties ("prefix": string, "namespace": anyURI)	
default namespace	anyURI	
base URI	anyURI	The static base URI of the XPath expression
expression	string	The XPath expression as a string

Schema (two arguments)

altova:schema(ComponentKind *as xs:string*, **Name** *as xs:QName*) **as (function(xs:string) as item()*)?** **XP3.1** **XQ3.1**

Returns the component kind that is specified in the first argument which has a name that is the same as the name supplied in the second argument. You can navigate further by selecting one of the component's properties.

- If this property is a component, you can navigate another step deeper by selecting one of this component's properties. This step can be repeated to navigate further into the schema.
- If the component is an atomic value, the atomic value is returned and you cannot navigate any deeper.

Note: In XQuery expressions, the schema must be explicitly imported. In XPath expressions, the schema must have been imported into the processing environment, for example, into XSLT with the **xslt:import** instruction.

Examples

- ```
  import schema "" at "C:\Test\ExpReport.xsd";
  altova:schema("element declaration", xs:QName("OrgChart"))("type definition")
  ("content type")("particles")[3]!.("term")("kind")
  ```
  returns the `kind` property of the term of the third `particles` component. This particles component is a descendant of the element declaration having a `QName` of `OrgChart`

- ```
  import schema "" at "C:\Test\ExpReport.xsd";
  let $typedef := altova:schema("type definition", xs:QName("emailType"))
  for $facet in $typedef ("facets")
  return [$facet ("kind"), $facet("value")]
  ```
  returns, for each **facet** of each **emailType** component, an array containing that facet's kind and value

Components and their properties

Assertion

Property name	Property type	Property value
kind	string	"Assertion"
test	XPath Property Record	

Attribute Declaration

Property name	Property type	Property value
kind	string	"Attribute Declaration"
name	string	Local name of the attribute
target namespace	string	Namespace URI of the attribute
type definition	Simple Type or Complex Type	
scope	A function with properties ("class":"Scope", "variety": "global" or "local", "parent": the containing Complex Type or Attribute Group)	
value constraint	If present, a function with properties ("class": "Value Constraint", "variety": "fixed" or "default", "value": atomic value, "lexical form": string. Note that the "value" property is not available for namespace-sensitive types	
inheritable	boolean	

Attribute Group Declaration

Property name	Property type	Property value
kind	string	"Attribute Group Definition"
name	string	Local name of the attribute group
target namespace	string	Namespace URI of the attribute group
attribute uses	Sequence of (Attribute Use)	
attribute wildcard	Optional Attribute Wildcard	

Attribute Use

Property name	Property type	Property value
kind	string	"Attribute Use"
required	boolean	true if the attribute is required, false if optional
value constraint	See Attribute Declaration	
inheritable	boolean	

 © 2017-2023 Altova GmbH

Attribute Wildcard

Property name	Property type	Property value
kind	string	"Wildcard"
namespace constraint	function with properties ("class": "Namespace Constraint", "variety": "any"|"enumeration"|"not", "namespaces": sequence of xs:anyURI, "disallowed names": list containing QNames and/or the strings "defined" and "definedSiblings"	
process contents	string ("strict"|"lax"|"skip")	

Complex Type

Property name	Property type	Property value
kind	string	"Complex Type"
name	string	Local name of the type (empty if anonymous)
target namespace	string	Namespace URI of the type (empty if anonymous)
base type definition	Complex Type Definition	
final	Sequence of strings ("restriction"|"extension")	
context	Empty sequence (not implemented)	
derivation method	string ("restriction"|"extension")	
abstract	boolean	
attribute uses	Sequence of Attribute Use	
attribute wildcard	Optional Attribute Wildcard	
content type	function with properties: ("class":"Content Type", "variety":string ("element-only"|"empty"|"mixed"|"simple"), particle: optional Particle, "open content": function with properties ("class":"Open Content", "mode": string ("interleave"|"suffix"), "wildcard": Wildcard), "simple type definition": Simple Type)	
prohibited substitutions	Sequence of strings ("restriction"|"extension")	
assertions	Sequence of Assertion	

Element Declaration

Property name	Property type	Property value
kind	string	"Complex Type"
name	string	Local name of the type (empty if anonymous)
target namespace	string	Namespace URI of the type (empty if anonymous)
type definition	Simple Type or Complex Type	
type table	function with properties ("class":"Type Table", "alternatives": sequence of Type Alternative, "default type definition": Simple Type or Complex Type)	
scope	function with properties ("class":"Scope", "variety": ("global"|"local"), "parent": optional Complex Type)	
value constraint	see Attribute Declaration	
nillable	boolean	
identity-constraint definitions	Sequence of Identity Constraint	
substitution group affiliations	Sequence of Element Declaration	
substitution group exclusions	Sequence of strings ("restriction"|"extension")	
disallowed substitutions	Sequence of strings ("restriction"|"extension"|"substitution")	
abstract	boolean	

Element Wildcard

Property name	Property type	Property value
kind	string	"Wildcard"
namespace constraint	function with properties ("class": "Namespace Constraint", "variety": "any"|"enumeration"|"not", "namespaces": sequence of xs:anyURI, "disallowed names": list containing QNames and/or the strings "defined" and "definedSiblings"	
process contents	string ("strict"|"lax"|"skip")	

Facet

Property name	Property type	Property value

 © 2017-2023 Altova GmbH

kind	string	The name of the facet, for example "minLength" or "enumeration"
value	depends on facet	The value of the facet
fixed	boolean	
typed-value	For the enumeration facet only, array(xs:anyAtomicType*)	An array containing the enumeration values, each of which may in general be a sequence of atomic values. (Note: for the enumeration facet, the "value" property is a sequence of strings, regardless of the actual type)

Identity Constraint

Property name	Property type	Property value
kind	string	"Identity-Constraint Definition"
name	string	Local name of the constraint
target namespace	string	Namespace URI of the constraint
identity-constraint category	string ("key"|"unique"|"keyRef")	
selector	XPath Property Record	
fields	Sequence of XPath Property Record	
referenced key	(For keyRef only): Identity Constraint	The corresponding key constraint

Model Group

Property name	Property type	Property value
kind	string	"Model Group"
compositor	string ("sequence"|"choice"|"all")	
particles	Sequence of Particle	

Model Group Definition

Property name	Property type	Property value
kind	string	"Model Group Definition"
name	string	Local name of the model group
target namespace	string	Namespace URI of the model group
model group	Model Group	

Notation

Property name	Property type	Property value
kind	string	"Notation Declaration"
name	string	Local name of the notation
target namespace	string	Namespace URI of the notation
system identifier	anyURI	
public identifier	string	

Particle

Property name	Property type	Property value
kind	string	"Particle"
min occurs	integer	
max occurs	integer, or string("unbounded")	
term	Element Declaration, Element Wildcard, or ModelGroup	

Simple Type

Property name	Property type	Property value
kind	string	"Simple Type Definition"
name	string	Local name of the type (empty if anonymous)
target namespace	string	Namespace URI of the type (empty if anonymous)
final	Sequence of string("restriction"|"extension"|"list"|"union")	
context	containing component	
base type definition	Simple Type	
facets	Sequence of Facet	
fundamental facets	Empty sequence (not implemented)	
variety	string ("atomic"|"list"|"union")	
primitive type definition	Simple Type	
item type definition	(for list types only) Simple Type	
member type definitions	(for union types only) Sequence of Simple Type	

Type Alternative

Property name	Property type	Property value

© 2017-2023 Altova GmbH

kind	string	"Type Alternative"
test	XPath Property Record	
type definition	Simple Type or Complex Type	

XPath Property Record

Property name	Property type	Property value
namespace bindings	Sequence of functions with properties ("prefix": string, "namespace": anyURI)	
default namespace	anyURI	
base URI	anyURI	The static base URI of the XPath expression
expression	string	The XPath expression as a string

Type

altova:type(Node *as item?***) as (function(xs:string) as item()*)? XP3.1 XQ3.1**
The function **altova:type** submits an element or attribute node of an XML document and returns the node's type information from the PSVI.

Note: The XML document must have a schema declaration so that the schema can be referenced.

Examples

- ```
  for $element in //Email
  let $type := altova:type($element)
  return $type
  ```
  returns a function that contains the `Email` node's type information

- ```
  for $element in //Email
  let $type := altova:type($element)
  return $type ("kind")
  ```
  takes the `Email` node's type component (Simple Type or Complex Type) and returns the value of the component's **kind** property

The **"_props"** parameter returns the properties of the selected component. For example:

- ```
  for $element in //Email
  let $type := altova:type($element)
  return ($type ("kind"), $type ("_props"))
  ```
  takes the `Email` node's type component (Simple Type or Complex Type) and returns (i) the value of the component's **kind** property, and then (ii) the properties of that component.

Components and their properties

Assertion
  ```

Property name	Property type	Property value
kind	string	"Assertion"
test	XPath Property Record	

Attribute Declaration

Property name	Property type	Property value
kind	string	"Attribute Declaration"
name	string	Local name of the attribute
target namespace	string	Namespace URI of the attribute
type definition	Simple Type or Complex Type	
scope	A function with properties ("class":"Scope", "variety": "global" or "local", "parent": the containing Complex Type or Attribute Group)	
value constraint	If present, a function with properties ("class": "Value Constraint", "variety": "fixed" or "default", "value": atomic value, "lexical form": string. Note that the "value" property is not available for namespace-sensitive types	
inheritable	boolean	

Attribute Group Declaration

Property name	Property type	Property value
kind	string	"Attribute Group Definition"
name	string	Local name of the attribute group
target namespace	string	Namespace URI of the attribute group
attribute uses	Sequence of (Attribute Use)	
attribute wildcard	Optional Attribute Wildcard	

Attribute Use

Property name	Property type	Property value
kind	string	"Attribute Use"
required	boolean	true if the attribute is required, false if optional
value constraint	See Attribute Declaration	
inheritable	boolean	

 © 2017-2023 Altova GmbH

Attribute Wildcard

Property name	Property type	Property value
kind	string	"Wildcard"
namespace constraint	function with properties ("class": "Namespace Constraint", "variety": "any"|"enumeration"|"not", "namespaces": sequence of xs:anyURI, "disallowed names": list containing QNames and/or the strings "defined" and "definedSiblings"	
process contents	string ("strict"|"lax"|"skip")	

Complex Type

Property name	Property type	Property value
kind	string	"Complex Type"
name	string	Local name of the type (empty if anonymous)
target namespace	string	Namespace URI of the type (empty if anonymous)
base type definition	Complex Type Definition	
final	Sequence of strings ("restriction"|"extension")	
context	Empty sequence (not implemented)	
derivation method	string ("restriction"|"extension")	
abstract	boolean	
attribute uses	Sequence of Attribute Use	
attribute wildcard	Optional Attribute Wildcard	
content type	function with properties: ("class":"Content Type", "variety":string ("element-only"|"empty"|"mixed"|"simple"), particle: optional Particle, "open content": function with properties ("class":"Open Content", "mode": string ("interleave"|"suffix"), "wildcard": Wildcard), "simple type definition": Simple Type)	
prohibited substitutions	Sequence of strings ("restriction"|"extension")	
assertions	Sequence of Assertion	

Element Declaration

Property name	Property type	Property value
kind	string	"Complex Type"
name	string	Local name of the type (empty if anonymous)
target namespace	string	Namespace URI of the type (empty if anonymous)
type definition	Simple Type or Complex Type	
type table	function with properties ("class":"Type Table", "alternatives": sequence of Type Alternative, "default type definition": Simple Type or Complex Type)	
scope	function with properties ("class":"Scope", "variety": ("global"|"local"), "parent": optional Complex Type)	
value constraint	see Attribute Declaration	
nillable	boolean	
identity-constraint definitions	Sequence of Identity Constraint	
substitution group affiliations	Sequence of Element Declaration	
substitution group exclusions	Sequence of strings ("restriction"|"extension")	
disallowed substitutions	Sequence of strings ("restriction"|"extension"|"substitution")	
abstract	boolean	

Element Wildcard

Property name	Property type	Property value
kind	string	"Wildcard"
namespace constraint	function with properties ("class": "Namespace Constraint", "variety": "any"|"enumeration"|"not", "namespaces": sequence of xs:anyURI, "disallowed names": list containing QNames and/or the strings "defined" and "definedSiblings"	
process contents	string ("strict"|"lax"|"skip")	

Facet

Property name	Property type	Property value
kind	string	The name of the facet, for

 © 2017-2023 Altova GmbH

		example "minLength" or "enumeration"
value	depends on facet	The value of the facet
fixed	boolean	
typed-value	For the enumeration facet only, array(xs:anyAtomicType*)	An array containing the enumeration values, each of which may in general be a sequence of atomic values. (Note: for the enumeration facet, the "value" property is a sequence of strings, regardless of the actual type)

Identity Constraint

Property name	Property type	Property value
kind	string	"Identity-Constraint Definition"
name	string	Local name of the constraint
target namespace	string	Namespace URI of the constraint
identity-constraint category	string ("key"|"unique"|"keyRef")	
selector	XPath Property Record	
fields	Sequence of XPath Property Record	
referenced key	(For keyRef only): Identity Constraint	The corresponding key constraint

Model Group

Property name	Property type	Property value
kind	string	"Model Group"
compositor	string ("sequence"|"choice"|"all")	
particles	Sequence of Particle	

Model Group Definition

Property name	Property type	Property value
kind	string	"Model Group Definition"
name	string	Local name of the model group
target namespace	string	Namespace URI of the model group
model group	Model Group	

Notation

Property name	Property type	Property value

kind	string	"Notation Declaration"
name	string	Local name of the notation
target namespace	string	Namespace URI of the notation
system identifier	anyURI	
public identifier	string	

Particle

Property name	Property type	Property value
kind	string	"Particle"
min occurs	integer	
max occurs	integer, or string("unbounded")	
term	Element Declaration, Element Wildcard, or ModelGroup	

Simple Type

Property name	Property type	Property value
kind	string	"Simple Type Definition"
name	string	Local name of the type (empty if anonymous)
target namespace	string	Namespace URI of the type (empty if anonymous)
final	Sequence of string("restriction"|"extension"|"list"|"union")	
context	containing component	
base type definition	Simple Type	
facets	Sequence of Facet	
fundamental facets	Empty sequence (not implemented)	
variety	string ("atomic"|"list"|"union")	
primitive type definition	Simple Type	
item type definition	(for list types only) Simple Type	
member type definitions	(for union types only) Sequence of Simple Type	

Type Alternative

Property name	Property type	Property value
kind	string	"Type Alternative"

© 2017-2023 Altova GmbH

test	XPath Property Record	
type definition	Simple Type or Complex Type	

XPath Property Record

Property name	Property type	Property value
namespace bindings	Sequence of functions with properties ("prefix": string, "namespace": anyURI)	
default namespace	anyURI	
base URI	anyURI	The static base URI of the XPath expression
expression	string	The XPath expression as a string

10.2.1.7 XPath/XQuery Functions: Sequence

Altova's sequence extension functions can be used in XPath and XQuery expressions and provide additional functionality for the processing of data. 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 XP3.1
XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3
XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3.1

attributes [altova:]

altova:attributes(AttributeName *as xs:string*) **as attribute()*** XP3.1 XQ3.1

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. This means that the context node must be the parent element node.

Examples

- **altova:attributes**("MyAttribute") returns MyAttribute()*

altova:attributes(AttributeName *as xs:string*, **SearchOptions** *as xs:string*) **as attribute()*** **XP3.1** **XQ3.1**

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 context node must be the parent element node. 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;
f = If this option is specified, then `AttributeName` provides a full match; otherwise `AttributeName` need only partially match an attribute name to return that attribute. For example: if **f** is not specified, then `MyAtt` will return `MyAttribute`;
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 as the second argument.

Examples

- **altova:attributes**("MyAttribute", "rfip") returns `MyAttribute()*`
- **altova:attributes**("MyAttribute", "pri") returns `MyAttribute()*`
- **altova:attributes**("MyAtt", "rip") returns `MyAttribute()*`
- **altova:attributes**("MyAttributes", "rfip") returns no match
- **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()*** **XP3.1** **XQ3.1**

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 context node must be the parent node of the element/s being searched for.

Examples

- **altova:elements**("MyElement") returns `MyElement()*`

altova:elements(ElementName *as xs:string*, **SearchOptions** *as xs:string*) **as element()*** **XP3.1** **XQ3.1**

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 context node must be the parent node of the element/s being searched for. 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;
f = If this option is specified, then `ElementName` provides a full match; otherwise `ElementName` need only partially match an element name to return that element. For example: if **f** is not specified, then `MyElem` will return `MyElement`;
i = switches to a case-insensitive search;
p = includes the namespace prefix in the search; `ElementName` should then contain the namespace prefix,

 © 2017-2023 Altova GmbH

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**("MyElem", "rip") returns MyElement()*
- **altova:elements**("MyElements", "rfip") returns no match
- **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()? XP3.1 XQ3.1**
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()*** **XP3.1** **XQ3.1**
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-combination**(11 to 20, 21 to 30, function($a, $b) {$a+$b = 32}) returns the sequence of xs:integers (11, 21)

 © 2017-2023 Altova GmbH

- `altova:find-first-combination(11 to 20, 21 to 30, function($a, $b) {$a+$b = 33})` returns the sequence of `xs:integers (11, 22)`
- `altova:find-first-combination(11 to 20, 21 to 30, function($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()*` XP3.1 XQ3.1
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` XP3.1 XQ3.1
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-pos**(11 to 20, 21 to 30, function($a, $b) {$a+$b = 32}) returns 1
- **altova:find-first-pair-pos**(11 to 20, 21 to 30, function($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** **XP3.1** **XQ3.1**

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:

 © 2017-2023 Altova GmbH

- **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.*

for-each-attribute-pair [altova:]

altova:for-each-attribute-pair(Seq1 *as element()?*, **Seq2** *as element()?*, **Function** *as function()*) **as item()*** **XP3.1** **XQ3.1**

The first two arguments identify two elements, the attributes of which are used to build attribute pairs, where one attribute of a pair is obtained from the first element and the other attribute is obtained from the second element. Attribute pairs are selected on the basis of having the same name, and the pairs are ordered alphabetically (on their names) into a set. If, for one attribute no corresponding attribute on the other element exists, then the pair is "disjoint", meaning that it consists of one member only. The function item (third argument Function) is applied separately to each pair in the sequence of pairs (joint and disjoint), resulting in an output that is a sequence of items.

Examples

- **altova:for-each-attribute-pair**(/Example/Test-A, /Example/Test-B, function($a, $b) {$a+b}) returns ...

`(2, 4, 6)` if
`<Test-A att1="1" att2="2" att3="3" />`
`<Test-B att1="1" att2="2" att3="3" />`

`(2, 4, 6)` if
`<Test-A att2="2" att1="1" att3="3" />`
`<Test-B att3="3" att2="2" att1="1" />`

`(2, 6)` if
`<Test-A att4="4" att1="1" att3="3" />`
`<Test-B att3="3" att2="2" att1="1" />`

Note: *The result* `(2, 6)` *is obtained by way of the following action:* `(1+1, ()+2, 3+3, 4+())`. *If one of the operands is the empty sequence, as in the case of items 2 and 4, then the result of the addition is an empty sequence.*

- **`altova:for-each-attribute-pair`**`(/Example/Test-A, /Example/Test-B, concat#2)` returns …

  `(11, 22, 33)` if
  `<Test-A att1="1" att2="2" att3="3" />`
  `<Test-B att1="1" att2="2" att3="3" />`

  `(11, 2, 33, 4)` if
  `<Test-A att4="4" att1="1" att3="3" />`
  `<Test-B att3="3" att2="2" att1="1" />`

▼ for-each-combination [altova:]

`altova:for-each-combination(FirstSequence` *`as item()*`*`, SecondSequence`** *`as item()*`*`, Function($i,$j){$i || $j} ) as item()*`** XP3.1 XQ3.1
The items of the two sequences in the first two arguments are combined so that each item of the first sequence is combined, in order, once with each item of the second sequence. The function given as the third argument is applied to each combination in the resulting sequence, resulting in an output that is a sequence of items (*see example*).

⊟ *Examples*

- **`altova:for-each-combination`**`( ('a', 'b', 'c'), ('1', '2', '3'), function($i, $j) {$i || $j} )` returns `('a1', 'a2', 'a3', 'b1', 'b2', 'b3', 'c1', 'c2', 'c3')`

▼ for-each-matching-attribute-pair [altova:]

`altova:for-each-matching-attribute-pair(Seq1` *`as element()?`*`, Seq2`** *`as element()?`*`, Function`** *`as function()`*`) as item()*`** XP3.1 XQ3.1
The first two arguments identify two elements, the attributes of which are used to build attribute pairs, where one attribute of a pair is obtained from the first element and the other attribute is obtained from the second element. Attribute pairs are selected on the basis of having the same name, and the pairs are ordered alphabetically (on their names) into a set. If, for one attribute no corresponding attribute on the other element exists, then no pair is built. The function item (third argument `Function`) is applied

 © 2017-2023 Altova GmbH

separately to each pair in the sequence of pairs, resulting in an output that is a sequence of items.

Examples

- `altova:for-each-matching-attribute-pair(/Example/Test-A, /Example/Test-B, function($a, $b){$a+b})` returns ...

  `(2, 4, 6)` if
  `<Test-A att1="1" att2="2" att3="3" />`
  `<Test-B att1="1" att2="2" att3="3" />`

  `(2, 4, 6)` if
  `<Test-A att2="2" att1="1" att3="3" />`
  `<Test-B att3="3" att2="2" att1="1" />`

  `(2, 6)` if
  `<Test-A att4="4" att1="1" att3="3" />`
  `<Test-B att3="3" att2="2" att3="1" />`

- `altova:for-each-matching-attribute-pair(/Example/Test-A, /Example/Test-B, concat#2)` returns ...

  `(11, 22, 33)` if
  `<Test-A att1="1" att2="2" att3="3" />`
  `<Test-B att1="1" att2="2" att3="3" />`

  `(11, 33)` if
  `<Test-A att4="4" att1="1" att3="3" />`
  `<Test-B att3="3" att2="2" att1="1" />`

▼ substitute-empty [altova:]

`altova:substitute-empty(FirstSequence` *`as item()*`*`, SecondSequence` *`as item()`*`) as item()*`

XP3.1 **XQ3.1**

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)`

10.2.1.8 XPath/XQuery Functions: String

Altova's string extension functions can be used in XPath and XQuery expressions and provide additional functionality for the processing of data. 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 XP3.1
XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3
XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3.1

camel-case [altova:]

altova:camel-case(InputString *as xs:string*) **as xs:string** XP3.1 XQ3.1
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** XP3.1 XQ3.1
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** XP3.1 XQ3.1
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**:

 © 2017-2023 Altova GmbH

- `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 XP3.1 XQ3.1**
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.

create-hash-from-string[altova:]

altova:create-hash-from-string(InputString *as xs:string*) **as xs:string XP2 XQ1 XP3.1 XQ3.1**
altova:create-hash-from-string(InputString *as xs:string*, **HashAlgo** *as xs:string*) **as xs:string XP2 XQ1 XP3.1 XQ3.1**
Generates a hash string from `InputString` by using the hashing algorithm specified by the `HashAlgo` argument. The following hashing algorithms may be specified (in upper or lower case): **MD5**, **SHA-1**, **SHA-224**, **SHA-256**, **SHA-384**, **SHA-512**. If the second argument is not specified (*see the first signature above*), then the **SHA-256** hashing algorithm is used.

Examples

- `altova:create-hash-from-string('abc')` returns a hash string generated by using the **SHA-256** hashing algorithm.
- `altova:create-hash-from-string('abc', 'md5')` returns a hash string generated by using the **MD5** hashing algorithm.
- `altova:create-hash-from-string('abc', 'MD5')` returns a hash string generated by using the **MD5** hashing algorithm.

first-chars [altova:]

altova:first-chars(X-Number *as xs:integer*) **as xs:string XP3.1 XQ3.1**
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 XP3.1 XQ3.1**

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`

format-string [altova:]

`altova:format-string(InputString` *`as xs:string`*`, FormatSequence` *`as item()*`*`) as xs:string` **XP3.1** **XQ3.1**

The input string (first argument) contains positional parameters (`%1`, `%2`, etc). Each parameter is replaced by the string item that is located at the corresponding position in the format sequence (submitted as the second argument). So the first item in the format sequence replaces the positional parameter `%1`, the second item replaces `%2`, and so on. The function returns this formatted string that contains the replacements. If no string exists for a positional parameter, then the positional parameter itself is returned. This happens when the index of a positional parameter is greater than the number of items in the format sequence.

Examples

- `altova:format-string('Hello %1, %2, %3', ('Jane','John','Joe'))` returns `"Hello Jane, John, Joe"`
- `altova:format-string('Hello %1, %2, %3', ('Jane','John','Joe', 'Tom'))` returns `"Hello Jane, John, Joe"`
- `altova:format-string('Hello %1, %2, %4', ('Jane','John','Joe', 'Tom'))` returns `"Hello Jane, John, Tom"`
- `altova:format-string('Hello %1, %2, %4', ('Jane','John','Joe'))` returns `"Hello Jane, John, %4"`

last-chars [altova:]

`altova:last-chars(X-Number` *`as xs:integer`*`) as xs:string` **XP3.1** **XQ3.1**

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` **XP3.1** **XQ3.1**

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`

© 2017-2023 Altova GmbH

pad-string-left [altova:]

altova:pad-string-left(StringToPad *as xs:string*, **StringLength** *as xs:integer*, **PadCharacter** *as xs:string*) **as xs:string** XP3.1 XQ3.1

The `PadCharacter` argument is a single character. It is padded to the left of the string to increase the number of characters in `StringToPad` so that this number equals the integer value of the `StringLength` argument. The `StringLength` argument can have any integer value (positive or negative), but padding will occur only if the value of `StringLength` is greater than the number of characters in `StringToPad`. If `StringToPad`. has more characters than the value of `StringLength`, then `StringToPad` is left unchanged.

Examples

- `altova:pad-string-left('AP', 1, 'Z')` returns `'AP'`
- `altova:pad-string-left('AP', 2, 'Z')` returns `'AP'`
- `altova:pad-string-left('AP', 3, 'Z')` returns `'ZAP'`
- `altova:pad-string-left('AP', 4, 'Z')` returns `'ZZAP'`
- `altova:pad-string-left('AP', -3, '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*, **StringLength** *as xs:integer*, **PadCharacter** *as xs:string*) **as xs:string** XP3.1 XQ3.1

The `PadCharacter` argument is a single character. It is padded to the right of the string to increase the number of characters in `StringToPad` so that this number equals the integer value of the `StringLength` argument. The `StringLength` argument can have any integer value (positive or negative), but padding will occur only if the value of `StringLength` is greater than the number of characters in `StringToPad`. If `StringToPad` has more characters than the value of `StringLength`, then `StringToPad` is left unchanged.

Examples

- `altova:pad-string-right('AP', 1, 'Z')` returns `'AP'`
- `altova:pad-string-right('AP', 2, 'Z')` returns `'AP'`
- `altova:pad-string-right('AP', 3, 'Z')` returns `'APZ'`
- `altova:pad-string-right('AP', 4, 'Z')` returns `'APZZ'`
- `altova:pad-string-right('AP', -3, '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 XP3.1 XQ3.1

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 XP3.1 XQ3.1

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** XP3.1 XQ3.1

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** XP3.1 XQ3.1

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** XP3.1 XQ3.1

© 2017-2023 Altova GmbH

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` XP3.1 XQ3.1

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` XP3.1 XQ3.1

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` XP3.1 XQ3.1

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"`

10.2.1.9 XPath/XQuery Functions: Miscellaneous

The following general purpose XPath/XQuery extension functions are supported in the current version of RaptorXML Server 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 XP3.1
XSLT functions (used in XPath expressions in XSLT):	XSLT1 XSLT2 XSLT3
XQuery functions (used in XQuery expressions in XQuery):	XQ1 XQ3.1

▼ decode-string [altova:]

altova:decode-string(Input *as xs:base64Binary*) **as xs:string** XP3.1 XQ3.1
altova:decode-string(Input *as xs:base64Binary*, **Encoding** *as xs:string*) **as xs:string** XP3.1 XQ3.1

Decodes the submitted base64Binary input to a string using the specified encoding. If no encoding is specified, then the UTF-8 encoding is used. The following encodings are supported: `US-ASCII`, `ISO-8859-1`, `UTF-16`, `UTF-16LE`, `UTF-16BE`, `ISO-10646-UCS2`, `UTF-32`, `UTF-32LE`, `UTF-32BE`, `ISO-10646-UCS4`

▣ *Examples*

- `altova:decode-string($XML1/MailData/Meta/b64B)` returns the base64Binary input as a UTF-8 encoded string
- `altova:decode-string($XML1/MailData/Meta/b64B, "UTF-8")` returns the base64Binary input as a UTF-8-encoded string
- `altova:decode-string($XML1/MailData/Meta/b64B, "ISO-8859-1")` returns the base64Binary input as an ISO-8859-1-encoded string

▼ encode-string [altova:]

© 2017-2023 Altova GmbH

altova:encode-string(InputString *as xs:string***) as xs:base64Binaryinteger** XP3.1 XQ3.1
altova:encode-string(InputString *as xs:string*, **Encoding** *as xs:string***) as xs:base64Binaryinteger** XP3.1 XQ3.1
Encodes the submitted string using, if one is given, the specified encoding. If no encoding is given, then the UTF-8 encoding is used. The encoded string is converted to base64Binary characters, and the converted base64Binary value is returned. Initially, UTF-8 encoding is supported, and support will be extended to the following encodings: `US-ASCII, ISO-8859-1, UTF-16, UTF-16LE, UTF-16BE, ISO-10646-UCS2, UTF-32, UTF-32LE, UTF-32BE, ISO-10646-UCS4`

Examples

- `altova:encode-string("Altova")` returns the base64Binary equivalent of the UTF-8 encoded string `"Altova"`
- `altova:encode-string("Altova", "UTF-8")` returns the base64Binary equivalent of the UTF-8 encoded string `"Altova"`

get-temp-folder [altova:]

altova:get-temp-folder() as `xs:string` XP2 XQ1 XP3.1 XQ3.1
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`.

generate-guid [altova:]

altova:generate-guid() as xs:string XP2 XQ1 XP3.1 XQ3.1
Generates a unique string GUID string.

Examples

- `altova:generate-guid()` returns (for example) `85F971DA-17F3-4E4E-994E-99137873ACCD`

high-res-timer [altova:]

altova:high-res-timer() as xs:double XP3.1 XQ3.1
Returns a system high-resolution timer value in seconds. A high-resolution timer, when present on a system, enables high precision time measurements when these are required (for example, in animations and for determining precise code-execution time). This function provides the resolution of the system's high-res timer.

Examples

- `altova:high-res-timer()` returns something like `'1.16766146154566E6'`

parse-html [altova:]

altova:parse-html(HTMLText *as xs:string***) as node()** XP3.1 XQ3.1
The `HTMLText` argument is a string that contains the text of an HTML document. The function creates an HTML tree from the string. The submitted string may or may not contain the HTML element. In either case, the root element of the tree is an element named **`HTML`**. It is best to make sure that the HTML code in the

submitted string is valid HTML.

Examples

- `altova:parse-html("<html><head/><body><h1>Header</h1></body></html>")` creates an HTML tree from the submitted string

sleep[altova:]

`altova:sleep(Millisecs` *`as xs:integer`*`) as empty-sequence()` XP2 XQ1 XP3.1 XQ3.1

Suspends execution of the current operation for the number of milliseconds given by the `Millisecs` argument.

Examples

- `altova:sleep(1000)` suspends execution of the current operation for 1000 milliseconds.

[Top 468]

10.2.1.10 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.

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 `x-binarytobase64` or `x-binarytobase16`

The function returns the chart image in the specified encoding.

`altova:generate-chart-image` `($chart, $width, $height, $encoding, $imagetype) as atomic`

 © 2017-2023 Altova GmbH

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 `x-binarytobase64` or `x-binarytobase16`
- `$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`. Alternatively to this function, you could also use the `xsl:result-document` function with `encoding="x-base64tobinary"`, where the image-data content is obtained via either the `generate-chart-image()` function or `chart()` function.

`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. Alternatively to this function, you could also use the `xsl:result-document` function with `encoding="x-base64tobinary"`, where the image-data content is obtained via either the `generate-chart-image()` function or `chart()` function.

Functions for creating charts

The following functions are used to create charts.

`altova:create-chart``($chart-config, $chart-data-series*) as chart extension item`

where

- `$chart-config` is the chart-config extension item obtained with the `altova:create-chart-config` function or via the `altova:create-chart-config-from-xml` function

- `$chart-data-series` is the chart-data-series extension item obtained with the `altova:create-chart-data-series` function or `altova:create-chart-data-series-from-rows` function

The function returns a chart extension item, which is created from the data supplied via the arguments.

altova:chart(`$chart-config, $chart-data-series*) as chart extension item`

where

- `$chart-config` is the chart-config extension item. It is an unordered series of four key: value pairs, where the four keys are `"width"`, `"height"`, `"title"`, and `"kind"`. The values of `width` and `height` are integers and specify the width and height of the chart in pixels. The value of `kind` is one of: `Pie, Pie3d, BarChart, BarChart3d, BarChart3dGrouped, LineChart, ValueLineChart, RoundGauge, BarGauge`.
- `$chart-data-series` is each an array of size 3, where each array defines a chart-data-series. Each array is composed of: (i) the name of the data series, (ii) the X-Axis values, (iii) the Y-Axis values. Multiple data series may be submitted; in the example below, for example, the two arrays respectively give data for monthly minimum and maximum temperatures.

The function returns an `xs:base64Binary` type item that contains the chart image. This image is created from the data supplied via the arguments of the function. Note that, since this function uses arrays and maps, it can be used only in XPath 3.1, XQuery 3.1, or XSLT 3.0.

Example:
```
altova:chart( map{'width':800, 'height':600, "kind":"LineChart", "title":"Monthly Temperatures"}, (['Min', $temps/Month, $temps/Month/@min], ['Max', $temps/Month, $temps/Month/@max]) )
```

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[474].

altova:create-chart-data-series(`$series-name?, $x-values*, $y-values*) as chart-data-series extension item`

where

© 2017-2023 Altova GmbH

- `$series-name` specifies the name of the series
- `$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

- `$chart-config` is the chart-config extension item obtained with the `altova:create-chart-config` function or via the `altova:create-chart-config-from-xml` function
- `$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
- `$chart-layer` is the chart-layer extension item obtained with the `altova:create-chart-layer` function
- `$mergecategoryvalues` merges the values of multiple data series if `true`, does not merge if `false`

The function returns a multi-layer-chart item.

10.2.1.10.1 Chart Data XML Structure

Given below is the XML structure of chart data, how it might appear for the Altova extension functions for charts [470]. 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
```


© 2017-2023 Altova GmbH

```
            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>
```

10.2.1.10.2 Example: Chart Functions

The example XSLT document below shows how Altova extension functions for charts 470 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=" ( (&amp;quot;Series 1&amp;quot;), &amp;apos;&amp;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>
```

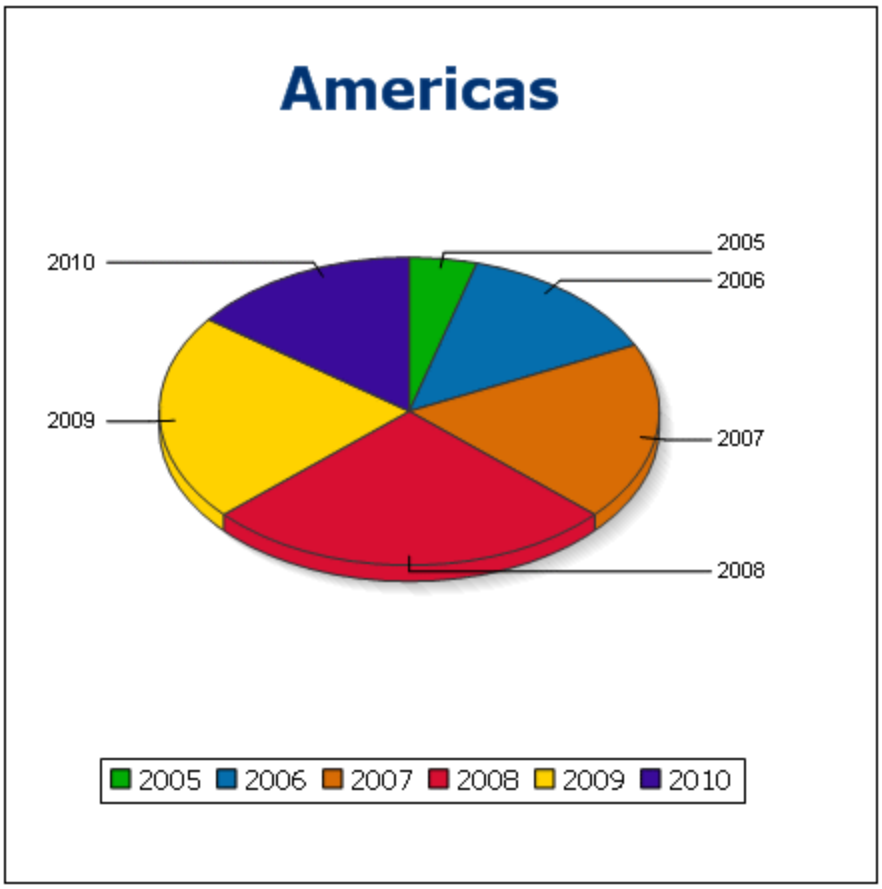
XML document

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.



10.2.1.11 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\Common2024\jar`.

The Java libraries used are in sub-folders of the folder *<ProgramFilesFolder>*`\Altova\Common2024\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.

© 2017-2023 Altova GmbH

Java virtual machine

In order to be able to use the barcode functions, a Java virtual machine must be available on your machine and it must match the bit version of the Altova application (32-bit or 64-bit). The path to the machine is found as noted below.

- If you are using an Altova desktop product, the Altova application attempts to detect the path to the Java virtual machine automatically, by reading (in this order): (i) the Windows registry, and (ii) the **JAVA_HOME** environment variable. You can also add a custom path in the Options dialog of the application; this entry will take priority over any other Java VM path detected automatically.
- If you are running an Altova server product on a Windows machine, the path to the Java virtual machine will be read first from the Windows registry; if this is not successful the **JAVA_HOME** environment variable will be used.
- If you are running an Altova server product on a Linux or macOS machine, then make sure that the **JAVA_HOME** environment variable is properly set and that the Java Virtual Machines library (on Windows, the **jvm.dll** file) can be located in either the **\bin\server** or **\bin\client** directory.

XSLT example to generate barcode

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="data:image/png;base64"/>
         </body>
      </html>
      <xsl:result-document
         href="barcode.png"
         method="text" encoding="base64tobinary" >
         <xsl:variable name="barcodeObject"
            select="altovaext-barcode:newInstance('Code39',string('some value'),
            96,0, (altovaext-barcode-property:new( 'setModuleWidth', 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>
```

XQuery example to generate QR code

Given below is an XQuery example showing how barcode functions can be used to generate a QR code image.

```
declare variable $lines := unparsed-text-
lines('https://info.healthministry.gv.at/data/timeline-cases-provinces.csv', 'utf-8');
declare variable $main := map:merge(tokenize(head($lines), ';')!map{.:position()});
declare variable $data := map:merge(tail($lines)!array{tokenize(., ';')}!map{?($main?Name):
[?($main?Date), xs:integer(?($main?ConfirmedCasesProvinces)) - xs:integer(?($main?
Recovered))]}, map{'duplicates':'combine'});
declare variable $chart_img := altovaext:chart(map{'width': 1900, 'height': 600}, map:for-
each($data, function($k, $v){[$k, $v?1!substring-before(., 'T'), $v?2][$k != 'Austria']}));
<img src="{$chart_img}"/>
(:$main,  $data,:)
```

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:

© 2017-2023 Altova GmbH

```
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
```

10.2.2 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[485] and .NET[494], as well as MSXSL scripts for XSLT[500]. 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[485]
- .NET Extension Functions[494]
- MSXSL Scripts for XSLT[500]

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.

10.2.2.1 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 491
- Java: Static Methods and Static Fields 491
- Java: Instance Methods and Instance Fields 492
- Datatypes: XPath/XQuery to Java 493
- Datatypes: Java to XPath/XQuery 494

Note the following

- If you are using an Altova desktop product, the Altova application attempts to detect the path to the Java virtual machine automatically, by reading (in this order): (i) the Windows registry, and (ii) the **JAVA_HOME** environment variable. You can also add a custom path in the Options dialog of the application; this entry will take priority over any other Java VM path detected automatically.
- If you are running an Altova server product on a Windows machine, the path to the Java virtual machine will be read first from the Windows registry; if this is not successful the **JAVA_HOME** environment variable will be used.
- If you are running an Altova server product on a Linux or macOS machine, then make sure that the **JAVA_HOME** environment variable is properly set and that the Java Virtual Machines library (on Windows, the **jvm.dll** file) can be located in either the **\bin\server** or **\bin\client** directory.

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)" />
```

 © 2017-2023 Altova GmbH

```
<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[487] and User-Defined Jar Files[490]. Note that paths to class files not in the current directory and to all JAR files must be specified.

10.2.2.1.1 User-Defined Class Files

If access is via a class file, then there are four 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*[488].)
- The class file is not packaged. The XSLT or XQuery file is in the same folder as the class file. (*See example below*[488].)
- The class file is in a package. The XSLT or XQuery file is at some random location. (*See example below*[488].)
- The class file is not packaged. The XSLT or XQuery file is at some random location. (*See example below*[489].)

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 referenced, XSLT/XQuery file in same folder as class file

The example below calls the `getVehicleType()` method of the `Car` class. Let us say that: (i) the `Car` class file is in the following folder: `JavaProject/com/altova/extfunc`, and (ii) that this folder is the current folder in the example below. 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

© 2017-2023 Altova GmbH

`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 referenced, XSLT/XQuery file at any location

The example below calls the `getCarColor()` method of the `Car` class. Let us say that the `Car` class file is in the folder `C:/JavaProject/com/altova/extfunc`, and the XSLT file is at any location. The location of the class file must then be 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.

10.2.2.1.2 User-Defined Jar Files

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

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

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

In the above:

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

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

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

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

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

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

<xsl:template match="/">
	<xsl:variable name="myCar" select="car:Car1.new('red')" />
	<a><xsl:value-of select="car:Car1.getCarColor($myCar)"/></a>
</xsl:template>

<xsl:template match="car"/>
```

 © 2017-2023 Altova GmbH

```
</xsl:stylesheet>
```

Note: When a path is supplied via the extension function, the path is added to the ClassLoader.

10.2.2.1.3 Java: Constructors

An extension function can be used to call a Java constructor. All constructors are called with the pseudo-function `new()`.

If the result of a Java constructor call can be implicitly converted to XPath/XQuery datatypes[494], 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*[492]):
  ```
  <xsl:value-of select="date:toString(date:new())" xmlns:date="java:java.util.Date" />
  ```

10.2.2.1.4 Java: Static Methods and Static Fields

A static method is called directly by its Java name and by supplying the arguments for the method. Static fields (methods that take no arguments), such as the constant-value fields `E` and `PI`, are accessed without specifying any argument.

XSLT examples

Here are some examples of how static methods and fields can be called:

```
<xsl:value-of xmlns:jMath="java:java.lang.Math"
              select="jMath:cos(3.14)" />

<xsl:value-of xmlns:jMath="java:java.lang.Math"
              select="jMath:cos( jMath:PI() )" />

<xsl:value-of xmlns:jMath="java:java.lang.Math"
              select="jMath:E() * jMath:cos(3.14)" />
```

Notice that the extension functions above have the form `prefix:fname()`. The prefix in all three cases is `jMath:`, which is associated with the namespace URI `java:java.lang.Math`. (The namespace URI must

begin with `java:`. In the examples above it is extended to contain the class name (`java.lang.Math`).) The `fname()` part of the extension functions must match the name of a public class (e.g. `java.lang.Math`) followed by the name of a public static method with its argument/s (such as `cos(3.14)`) or a public static field (such as `PI()`).

In the examples above, the class name has been included in the namespace URI. If it were not contained in the namespace URI, then it would have to be included in the `fname()` part of the extension function. For example:

```
<xsl:value-of xmlns:java="java:"
              select="java:java.lang.Math.cos(3.14)" />
```

XQuery example

A similar example in XQuery would be:

```
<cosine xmlns:jMath="java:java.lang.Math">
   {jMath:cos(3.14)}
</cosine>
```

10.2.2.1.5 Java: Instance Methods and Instance Fields

An instance method has a Java object passed to it as the first argument of the method call. Such a Java object typically would be created by using an extension function (for example a constructor call) or a stylesheet parameter/variable. An XSLT example of this kind would be:

```
<xsl:stylesheet version="1.0" exclude-result-prefixes="date"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:date="java:java.util.Date"
  xmlns:jlang="java:java.lang">
  <xsl:param name="CurrentDate" select="date:new()"/>
  <xsl:template match="/">
      <enrollment institution-id="Altova School"
                date="{date:toString($CurrentDate)}"
                type="{jlang:Object.toString(jlang:Object.getClass( date:new() ))}">
      </enrollment>
  </xsl:template>
</xsl:stylesheet>
```

In the example above, the value of the node `enrollment/@type` is created as follows:

1. An object is created with a constructor for the class `java.util.Date` (with the `date:new()` constructor).
2. This Java object is passed as the argument of the `jlang.Object.getClass` method.
3. The object obtained by the `getClass` method is passed as the argument to the `jlang.Object.toString` method.

The result (the value of `@type`) will be a string having the value: `java.util.Date`.

An instance field is theoretically different from an instance method in that it is not a Java object per se that is passed as an argument to the instance field. Instead, a parameter or variable is passed as the argument. However, the parameter/variable may itself contain the value returned by a Java object. For example, the

 © 2017-2023 Altova GmbH

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`.

10.2.2.1.6 Datatypes: XPath/XQuery to Java

When a Java function is called from within an XPath/XQuery expression, the datatype of the function's arguments is important in determining which of multiple Java classes having the same name is called.

In Java, the following rules are followed:

- If there is more than one Java method with the same name, but each has a different number of arguments than the other/s, then the Java method that best matches the number of arguments in the function call is selected.
- The XPath/XQuery string, number, and boolean datatypes (*see list below*) are implicitly converted to a corresponding Java datatype. If the supplied XPath/XQuery type can be converted to more than one Java type (for example, `xs:integer`), then that Java type is selected which is declared for the selected method. For example, if the Java method being called is `fx(decimal)` and the supplied XPath/XQuery datatype is `xs:integer`, then `xs:integer` will be converted to Java's `decimal` datatype.

The table below lists the implicit conversions of XPath/XQuery string, number, and boolean types to Java datatypes.

`xs:string`	`java.lang.String`
`xs:boolean`	`boolean` (primitive), `java.lang.Boolean`
`xs:integer`	int, long, short, byte, float, double, and the wrapper classes of these, such as `java.lang.Integer`
`xs:float`	`float` (primitive), `java.lang.Float`, `double` (primitive)
`xs:double`	`double` (primitive), `java.lang.Double`
`xs:decimal`	`float` (primitive), `java.lang.Float`, `double`(primitive), `java.lang.Double`

Subtypes of the XML Schema datatypes listed above (and which are used in XPath and XQuery) will also be converted to the Java type/s corresponding to that subtype's ancestor type.

In some cases, it might not be possible to select the correct Java method based on the supplied information. For example, consider the following case.

- The supplied argument is an `xs:untypedAtomic` value of 10 and it is intended for the method `mymethod(float)`.
- However, there is another method in the class which takes an argument of another datatype: `mymethod(double)`.

- Since the method names are the same and the supplied type (`xs:untypedAtomic`) could be converted correctly to either `float` or `double`, it is possible that `xs:untypedAtomic` is converted to `double` instead of `float`.
- Consequently the method selected will not be the required method and might not produce the expected result. To work around this, you can create a user-defined method with a different name and use this method.

Types that are not covered in the list above (for example `xs:date`) will not be converted and will generate an error. However, note that in some cases, it might be possible to create the required Java type by using a Java constructor.

10.2.2.1.7 Datatypes: Java to XPath/XQuery

When a Java method returns a value, the datatype of the value is a string, numeric or boolean type, then it is converted to the corresponding XPath/XQuery type. For example, Java's `java.lang.Boolean` and `boolean` datatypes are converted to `xsd:boolean`.

One-dimensional arrays returned by functions are expanded to a sequence. Multi-dimensional arrays will not be converted, and should therefore be wrapped.

When a wrapped Java object or a datatype other than string, numeric or boolean is returned, you can ensure conversion to the required XPath/XQuery type by first using a Java method (e.g `toString`) to convert the Java object to a string. In XPath/XQuery, the string can be modified to fit the lexical representation of the required type and then converted to the required type (for example, by using the `cast as` expression).

10.2.2.2 .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 496
- .NET: Static Methods and Static Fields 497
- .NET: Instance Methods and Instance Fields 498
- Datatypes: XPath/XQuery to .NET 499
- Datatypes: .NET to XPath/XQuery 500

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.

 © 2017-2023 Altova GmbH

- 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.

10.2.2.2.1 .NET: Constructors

An extension function can be used to call a .NET constructor. All constructors are called with the pseudo-function `new()`. If there is more than one constructor for a class, then the constructor that most closely matches the number of arguments supplied is selected. If no constructor is deemed to match the supplied argument/s, then a `'No constructor found'` error is returned.

Constructors that return XPath/XQuery datatypes

If the result of a .NET constructor call can be implicitly converted to XPath/XQuery datatypes 494, then the .NET extension function will return a sequence that is an XPath/XQuery datatype.

 © 2017-2023 Altova GmbH

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* 492):
  ```
  <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(date:get_Month(date:new(2008, 4, 29)))"
  xmlns:date="clitype:System.DateTime" />
  ```

10.2.2.2.2 .NET: Static Methods and Static Fields

A static method is called directly by its name and by supplying the arguments for the method. The name used in the call must exactly match a public static method in the class specified. If the method name and the number of arguments that were given in the function call matches more than one method in a class, then the types of the supplied arguments are evaluated for the best match. If a match cannot be found unambiguously, an error is reported.

Note: A field in a .NET class is considered to be a method without any argument. A property is called using the syntax `get_PropertyName()`.

Examples

An XSLT example showing a call to a method with one argument (`System.Math.Sin(arg)`):

```
<xsl:value-of select="math:Sin(30)" xmlns:math="clitype:System.Math"/>
```

An XSLT example showing a call to a field (considered a method with no argument) (`System.Double.MaxValue()`):

```
<xsl:value-of select="double:MaxValue()" xmlns:double="clitype:System.Double"/>
```

An XSLT example showing a call to a property (syntax is `get_PropertyName()`) (`System.String()`):

```
<xsl:value-of select="string:get_Length('my string')"
xmlns:string="clitype:System.String"/>
```
```

An XQuery example showing a call to a method with one argument (`System.Math.Sin(arg)`):

```
<sin xmlns:math="clitype:System.Math">
   { math:Sin(30) }
</sin>
```

10.2.2.2.3 .NET: Instance Methods and Instance Fields

An instance method has a .NET object passed to it as the first argument of the method call. This .NET object typically would be created by using an extension function (for example a constructor call) or a stylesheet parameter/variable. An XSLT example of this kind would be:

```
<xsl:stylesheet version="2.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:fn="http://www.w3.org/2005/xpath-functions">
   <xsl:output method="xml" omit-xml-declaration="yes"/>
   <xsl:template match="/">
      <xsl:variable name="releasedate"
         select="date:new(2008, 4, 29)"
         xmlns:date="clitype:System.DateTime"/>
      <doc>
         <date>
            <xsl:value-of select="date:ToString(date:new(2008, 4, 29))"
               xmlns:date="clitype:System.DateTime"/>
         </date>
         <date>
            <xsl:value-of select="date:ToString($releasedate)"
               xmlns:date="clitype:System.DateTime"/>
         </date>
      </doc>
   </xsl:template>
</xsl:stylesheet>
```

In the example above, a `System.DateTime` constructor (`new(2008, 4, 29)`) is used to create a .NET object of type `System.DateTime`. This object is created twice, once as the value of the variable `releasedate`, a second time as the first and only argument of the `System.DateTime.ToString()` method. The instance method `System.DateTime.ToString()` is called twice, both times with the `System.DateTime` constructor (`new(2008, 4, 29)`) as its first and only argument. In one of these instances, the variable `releasedate` is used to get the .NET object.

Instance methods and instance fields

The difference between an instance method and an instance field is theoretical. In an instance method, a .NET object is directly passed as an argument; in an instance field, a parameter or variable is passed instead—though the parameter or variable may itself contain a .NET object. For example, in the example above, the variable `releasedate` contains a .NET object, and it is this variable that is passed as the argument of `ToString()` in the second `date` element constructor. Therefore, the `ToString()` instance in the first `date` element is an instance method while the second is considered to be an instance field. The result produced in both instances, however, is the same.

 © 2017-2023 Altova GmbH

10.2.2.2.4 Datatypes: XPath/XQuery to .NET

When a .NET extension function is used within an XPath/XQuery expression, the datatypes of the function's arguments are important for determining which one of multiple .NET methods having the same name is called.

In .NET, the following rules are followed:

- If there is more than one method with the same name in a class, then the methods available for selection are reduced to those that have the same number of arguments as the function call.
- The XPath/XQuery string, number, and boolean datatypes (*see list below*) are implicitly converted to a corresponding .NET datatype. If the supplied XPath/XQuery type can be converted to more than one .NET type (for example, `xs:integer`), then that .NET type is selected which is declared for the selected method. For example, if the .NET method being called is `fx(double)` and the supplied XPath/XQuery datatype is `xs:integer`, then `xs:integer` will be converted to .NET's `double` datatype.

The table below lists the implicit conversions of XPath/XQuery string, number, and boolean types to .NET datatypes.

`xs:string`	`StringValue, string`
`xs:boolean`	`BooleanValue, bool`
`xs:integer`	`IntegerValue, decimal, long, integer, short, byte, double, float`
`xs:float`	`FloatValue, float, double`
`xs:double`	`DoubleValue, double`
`xs:decimal`	`DecimalValue, decimal, double, float`

Subtypes of the XML Schema datatypes listed above (and which are used in XPath and XQuery) will also be converted to the .NET type/s corresponding to that subtype's ancestor type.

In some cases, it might not be possible to select the correct .NET method based on the supplied information. For example, consider the following case.

- The supplied argument is an `xs:untypedAtomic` value of 10 and it is intended for the method `mymethod(float)`.
- However, there is another method in the class which takes an argument of another datatype: `mymethod(double)`.
- Since the method names are the same and the supplied type (`xs:untypedAtomic`) could be converted correctly to either `float` or `double`, it is possible that `xs:untypedAtomic` is converted to `double` instead of `float`.
- Consequently the method selected will not be the required method and might not produce the expected result. To work around this, you can create a user-defined method with a different name and use this method.

Types that are not covered in the list above (for example `xs:date`) will not be converted and will generate an error.

10.2.2.2.5 Datatypes: .NET to XPath/XQuery

When a .NET method returns a value and the datatype of the value is a string, numeric or boolean type, then it is converted to the corresponding XPath/XQuery type. For example, .NET's `decimal` datatype is converted to `xsd:decimal`.

When a .NET object or a datatype other than string, numeric or boolean is returned, you can ensure conversion to the required XPath/XQuery type by first using a .NET method (for example `System.DateTime.ToString()`) to convert the .NET object to a string. In XPath/XQuery, the string can be modified to fit the lexical representation of the required type and then converted to the required type (for example, by using the `cast as` expression).

10.2.2.3 MSXSL Scripts for XSLT

The `<msxsl:script>` element contains user-defined functions and variables that can be called from within XPath expressions in the XSLT stylesheet. The `<msxsl:script>` is a top-level element, that is, it must be a child element of `<xsl:stylesheet>` or `<xsl:transform>`.

The `<msxsl:script>` element must be in the namespace `urn:schemas-microsoft-com:xslt` (*see example below*).

Scripting language and namespace

The scripting language used within the block is specified in the `<msxsl:script>` element's `language` attribute and the namespace to be used for function calls from XPath expressions is identified with the `implements-prefix` attribute (*see below*).

```
<msxsl:script language="scripting-language" implements-prefix="user-namespace-prefix">

   function-1 or variable-1
   ...
   function-n or variable-n

</msxsl:script>
```

The `<msxsl:script>` element interacts with the Windows Scripting Runtime, so only languages that are installed on your machine may be used within the `<msxsl:script>` element. **The .NET Framework 2.0 platform or higher must be installed for MSXSL scripts to be used.** Consequently, the .NET scripting languages can be used within the `<msxsl:script>` element.

The `language` attribute accepts the same values as the `language` attribute on the HTML `<script>` element. If the `language` attribute is not specified, then Microsoft JScript is assumed as the default.

The implements-prefix attribute takes a value that is a prefix of a declared in-scope namespace. This namespace typically will be a user namespace that has been reserved for a function library. All functions and

 © 2017-2023 Altova GmbH

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.

© 2017-2023 Altova GmbH

Index

.

.NET extension functions,
- constructors, 496
- datatype conversions (.NET to XPath/XQuery), 500
- datatype conversions (XPath/XQuery to .NET), 499
- for XSLT and XQuery, 494
- instance methods, instance fields, 498
- overview, 494
- static methods, static fields, 497

.NET Framework API, 362
.NET interface, 11

A

Altova extensions,
- chart functions (see chart functions), 394

Altova ServiceController, 29
Assgning a license to RaptorXML Server on Linux, 37
Assgning a license to RaptorXML Server on macOS, 42
Assgning a license to RaptorXML Server on Windows, 31

C

C# example for REST API, 271
Catalog customization, 50
Catalog mechanism overview, 47
Catalogs, 47
Catalogs and envirnment variables, 52
Catalogs in RaptorXML, 48
Chart functions,
- chart data structure for, 474
- example, 479
- listing, 470

COM interface, 11
Comman line,
- options, 221

Command line,
- and XQuery, 92
- usage summary, 56

CoreCatalog.xml, 48
CustomCatalog.xml, 48

D

Debugging Python scripts in Visual Studio code, 358
Debugging server-side Python scripts, 358
Deinstallation, 22
DTDs and catalogs, 47

E

Environment variables used in catalogs, 48
Environnment variables, 52
Extension functions for XSLT and XQuery, 485
Extension Functions in .NET for XSLT and XQuery,
- see under .NET extension functions, 494

Extension Functions in Java for XSLT and XQuery,
- see under Java extension functions, 485

Extension Functions in MSXSL scripts, 500

G

Global resources, 53

H

Help command on CLI, 197
HTTP interface, 11, 238
- client requests, 250
- example project, 270
- security issues, 55
- server configuration, 242
- server setup, 239

I

Installation of RaptorXML Server, 21

© 2017-2023 Altova GmbH

Installation on Linux, 33
Installation on macOS, 39
Installing LicenseServer on Linux, 35
Installing LicenseServer on macOS, 40
Installing LicenseServer on Windows, 26
Installing on Windows, 22
Installing on Windows Server Core, 23
service properties, 26
SSL webserver properties, 25
webserver properties, 25
installing RaptorXMLServer Python module, 355
Interfaces,
overview of, 11

J

Java extension functions,
constructors, 491
datatype conversions, Java to Xpath/XQuery, 494
datatype conversions, XPath/XQuery to Java, 493
for XSLT and XQuery, 485
instance methods, instance fields, 492
overview, 485
static methods, static fields, 491
user-defined class files, 487
user-defined JAR files, 490
Java interface, 11
JSON config file,
for RaptorXMLServer Python module, 355

L

License commands on CLI, 202
License for RaptorXML Server,
assigning on Linux, 37
assigning on macOS, 42
assigning on Windows, 31
LicenseServer versions, 26, 35, 40
Licensing of RaptorXML Server, 21
Licensing RaptorXML Server on Linux, 36
Licensing RaptorXML Server on macOS, 41
Licensing RaptorXML Server on Windows, 28
Linux,
installation on, 33
licensing RaptorXML Server on, 36
Localization, 199

M

macOS,
installation on, 39
licensing RaptorXML Server on, 41
Migrating RaptorXML Server to a new machine, 45
msxsl:script, 500

N

Network connections, 27

P

pip command, 355
Python,
security issues, 55
Python API, 352
Python API FAQs, 360
Python interface, 11
Python library,
of RaptorXML Server, 355
Python module,
of RaptorXML Server, 355
Python scripts on RaptorXML Server, 358

R

RaptorXML,
command line interface, 11
editions and interfaces, 11
features, 16
HTTP interface, 11
interfaces with COM, Java, .NET, 11
introduction, 10
Python interface, 11
supported specifications, 18
system requirements, 15

© 2017-2023 Altova GmbH

RaptorXML Server,
migrating to a new machine, 45
RaptorXML Server APIs, 351
Register RaptorXML Server with LicenseServer on Linux, 37
Register RaptorXML Server with LicenseServer on macOS, 42
Register RaptorXML Server with LicenseServer on Windows, 31
REST API,
example project, 270
REST interface,
wrapper class, 270
RootCatalog.xml, 48, 355

S

Schema Manager,
CLI Help command, 375
CLI Info command, 376
CLI Initialize command, 376
CLI Install command, 377
CLI List command, 377
CLI overview, 375
CLI Reset command, 378
CLI Uninstall command, 379
CLI Update command, 380
CLI Upgrade command, 380
how to run, 367
installing a schema, 372
listing schemas by status in, 370
overview of, 363
patching a schema, 372
resetting, 374
status of schemas in, 370
uninstalling a achema, 374
upgrading a schema, 372
Schemas,
looking up via catalogs, 50
Schemas and catalogs, 47
Scripts in XSLT/XQuery,
see under Extension functions, 485
Security issues, 55
Server configuration, 242
Service configuration, 27
Setup,
on Linux, 33
on macOS, 39
on Windows, 22
Setup of RaptorXML Server, 21
Start LicenseServer on Linux, 36
Start LicenseServer on macOS, 41
Start LicenseServer on Windows, 29
Start RaptorXML Server on Linux, 36
Start RaptorXML Server on macOS, 41
Start RaptorXML Server on Windows, 29

U

Uninstalling, 22
Upgrading RaptorXML Server on Windows, 44

V

Validation,
of DTD, 69
of XML instance with DTD, 58
of XML instance with XSD, 62
of XQuery document, 108
of XSD, 73
of XSLT document, 129
Visua Studio and Python scripts, 358

W

Well-formedness check, 80
Windows,
installation on, 22
licensing RaptorXML Server on, 28
upgrading RaptorXML Server on, 44
Wrapper class for REST interface, 270

X

XML catalogs, 47
XQuery,
Extension functions, 485
XQuery commands, 92

© 2017-2023 Altova GmbH

XQuery document validation, 108
XQuery execution, 92
XSLT,
Extension functions, 485
XSLT commands, 121
XSLT document validation, 129
XSLT transformation, 121

© 2017-2023 Altova GmbH